MODÉLISATION FORMELLE DES SYSTÈMES DE DÉTECTION D'INTRUSIONS

par

Lionel NGANYEWOU TIDJON

Thèse en cotutelle présentée au Département d'informatique
en vue de l'obtention du grade de philosophiæ doctor (Ph.D.)

FACULTÉ DES SCIENCES

UNIVERSITÉ DE SHERBROOKE

Sherbrooke, Québec, Canada, 2020-10-12

Le 2020-10-12

Le jury a accepté la thèse de Lionel NGANYEWOU TIDJON dans sa version finale

Membres du jury

Professeur Gabriel Girard
Président-rapporteur
Département Informatique
Université de Sherbrooke, Canada

Professeur Yamine AIT AMEUR
Rapporteur
Département Informatique
INP-ENSEEIHT, Toulouse, France

Professeur Sylvain HALLE
Rapporteur
Département Informatique
Université du Québec à Chicoutimi, Canada

Professeur Marc FRAPPIER
Codirecteur
Département Informatique
Université de Sherbrooke, Canada

Professeur Amel MAMMAR
Codirectrice
Département Informatique
Télécom-SudParis, Institut Polytechnique de Paris, France

Professeur Joaquin GÀRCIA-ALFARO
Examinateur
Département Informatique
Télécom-SudParis, Institut Polytechnique de Paris, France

Sommaire

L'écosystème de la cybersécurité évolue en permanence en termes du nombre, de la diversité, et de la complexité des attaques. De ce fait, les outils de détection deviennent inefficaces face à certaines attaques. On distingue généralement trois types de systèmes de détection d'intrusions : détection par anomalies, détection par signatures et détection hybride. La détection par anomalies est fondée sur la caractérisation du comportement *habituel* du système, typiquement de manière statistique. Elle permet de détecter des attaques connues ou inconnues, mais génère aussi un très grand nombre de faux positifs. La détection par signatures permet de détecter des attaques connues en définissant des règles qui décrivent le comportement connu d'un attaquant. Cela demande une bonne connaissance du comportement de l'attaquant. La détection hybride repose sur plusieurs méthodes de détection incluant celles sus-citées. Elle présente l'avantage d'être plus précise pendant la détection. Des outils tels que Snort et Zeek offrent des langages de bas niveau pour l'expression de règles de reconnaissance d'attaques. Le nombre d'attaques potentielles étant très grand, ces bases de règles deviennent rapidement difficiles à gérer et à maintenir. De plus, l'expression de règles avec état dit *stateful* est particulièrement ardue pour reconnaître une séquence d'événements.

Dans cette thèse, nous proposons une approche *stateful* basée sur les diagrammes d'état-transition algébriques (ASTDs) afin d'identifier des attaques complexes. Les ASTDs permettent de représenter de façon graphique et modulaire une spécification, ce qui facilite la maintenance et la compréhension des règles. Nous étendons la notation ASTD avec de nouvelles fonctionnalités pour représenter des attaques complexes. Ensuite, nous spécifions plusieurs attaques avec la notation étendue et exécutons les spécifications obtenues sur des flots d'événements à l'aide d'un interpréteur pour iden-

tifier des attaques. Nous évaluons aussi les performances de l'interpréteur avec des outils industriels tels que Snort et Zeek. Puis, nous réalisons un compilateur afin de générer du code exécutable à partir d'une spécification ASTD, capable d'identifier de façon efficiente les séquences d'événements.

Mots-clés: Détection d'intrusions, Spécification formelle, Compilation, Preuves, ASTD

Abstract

The cybersecurity ecosystem continuously evolves with the number, the diversity, and the complexity of cyber attacks. Generally, we have three types of Intrusion Detection System (IDS) : anomaly-based detection, signature-based detection, and hybrid detection. Anomaly detection is based on the *usual* behavior description of the system, typically in a static manner. It enables detecting known or unknown attacks but also generating a large number of false positives. Signature based detection enables detecting known attacks by defining rules that describe known attacker's behavior. It needs a good knowledge of attacker behavior. Hybrid detection relies on several detection methods including the previous ones. It has the advantage of being more precise during detection. Tools like Snort and Zeek offer low level languages to represent rules for detecting attacks. The number of potential attacks being large, these rule bases become quickly hard to manage and maintain. Moreover, the representation of *stateful* rules to recognize a sequence of events is particularly arduous.

In this thesis, we propose a *stateful* approach based on algebraic state-transition diagrams (ASTDs) to identify complex attacks. ASTDs allow a graphical and modular representation of a specification, that facilitates maintenance and understanding of rules. We extend the ASTD notation with new features to represent complex attacks. Next, we specify several attacks with the extended notation and run the resulting specifications on event streams using an interpreter to identify attacks. We also evaluate the performance of the interpreter with industrial tools such as Snort and Zeek. Then, we build a compiler in order to generate executable code from an ASTD specification, able to efficiently identify sequences of events.

Keywords: Intrusion detection, Formal spécification, Compilation, Proofs, ASTD

Remerciements

Je tiens à remercier mes directeurs de thèse, Pr Marc Frappier et Pre Amel Mammar, pour leur pragmatisme, leurs conseils et encouragements durant le cursus doctoral.

Merci aux membres du jury pour le grand honneur qu'ils nous ont fait en acceptant de juger ce travail.

Je remercie également l'équipe administrative de l'Université de Sherbrooke et de Télécom SudParis pour l'assistance durant le processus de cotutelle de thèse et de partenariats avec des organismes privés et gouvernementaux.

Merci à ma maman, ma bien-aimée, mes grands frères et sœurs, ainsi qu'à tous mes amis qui m'ont encouragé durant les moments difficiles tout au long de ma thèse.

Abréviations

ASTD Algebraic State Transition Diagram

ASM Abstract State Machine

ATT&CK Adversarial Tactics, Techniques, and Common Knowledge

APT Advanced Persistent Threat

CAPEC Common Attack Pattern Enumeration and Classification

CSP Communicating Sequential Processes

CCS Calculus of Communicating Systems

CybOX Cyber Observable eXpression

CVE Common Vulnerability Enumeration

CPE Common Platform Enumeration

CWE Common Weaknesses Enumeration

ESP Event Stream Processing

IDS Intrusion Detection System

IPS Intrusion Prevention System

MAEC Malware Attribute Enumeration and Characterization

MANET Mobile Ad-Hoc Network

SIEM Security Information and Event Management

STATL State-Transition-based Attack Language

SOC Security Operations Center

STIX Structured Threat Information eXpression

WSN Wireless Sensor Network

Table des matières

Liste des figures

Liste des tableaux

Introduction

En 2016, les dégâts causés par les attaques cybernétiques étaient évalués à plus de 450 milliards de dollars [121]. Malgré les efforts pour y remédier, les attaques continuent de croître rapidement, touchant les systèmes d'information, les infrastructures industrielles, les réseaux informatiques et les appareils personnels. Dans la majorité des cas, les causes majeures des attaques sont les vulnérabilités liées aux défauts de conception et erreurs humaines telles que : l'ouverture de courriels malicieux, consultation de pages Web non sécurisées et le manque de mises à jour du système d'exploitation. De nombreux outils de prévention et de détection d'attaques ont été proposés afin d'assurer le contrôle, précisément la sécurité des données sensibles et la prise de décision sur les comportements d'attaques observés. Les outils sont placés stratégiquement dans un réseau ou un hôte, où ils opèrent en mode singulier, distribué, centralisé, collaboratif ou coopératif afin de prévenir et/ou détecter des attaques visant la confidentialité, l'intégrité ou la disponibilité des informations. La précision et la performance de tels outils dépendent de plusieurs critères parmi lesquels : la méthode de détection, le type de réponse, l'environnement et le type de système de détection d'intrusions.

Les méthodes de détection peuvent être regroupées en trois groupes : la détection par signatures, la détection par anomalies et la détection hybride. La détection par signatures utilise des règles ou des motifs prédéfinis afin d'identifier des attaques conformément à ces motifs. Tandis que la détection par anomalies se base sur des profils statistiques représentatifs du comportement normal du système et toute déviation à ces profils (comportement anormal) est considérée comme une attaque. La méthode hybride exploite les deux méthodes sus-citées afin d'améliorer la détection. Actuellement, les analystes en cybersécurité combinent la détection par anomalies

et par signatures afin d'optimiser le taux de détection. En effet, la détection par signatures est appropriée pour les attaques connues, c'est-à-dire qui existent dans sa base de signatures, alors que la détection par anomalies est efficace pour identifier les attaques inconnues. Face à un volume de données élevé, les centres des opérations en sécurité (SOCs) associent les outils basés sur les méthodes sus-citées avec un système de gestion des événements de sécurité afin d'avoir une vue holistique des postures de sécurité de l'organisation.

Problématique

Les attaques cybernétiques évoluent très rapidement, étant de plus en plus non prévisibles et persistantes. De ce fait, les outils de détection d'attaques deviennent inefficaces face à certaines attaques. Les outils tels que Snort [271] et Zeek [246] nécessitent une mise à jour des règles de détection à chaque fois qu'une attaque inconnue est observée. En général, cette mise à jour se fait manuellement par un expert en cybersécurité dû à la nature complexe des comportements d'attaques. De plus, les règles sont exprimées dans un langage de bas niveau limitant la reconnaissance de nouvelles d'attaques i.e., elles sont essentiellement exprimées en utilisant des fonctions et des variables globales (cas de Zeek) ou représentées dans un format ASCII difficile à comprendre (cas de Snort). En effet, les langages utilisés pour l'expression des règles représentent celles-ci dans des environnements particuliers et souffrent des idiosyncrasies de leurs paramètres opérationnels qui les rendent difficiles à étendre à de nouveaux environnements. Ainsi, le modèle sémantique ou les abstractions fournis au développeur des règles sont soit de bas niveau, soit pas clairement définis. D'autre part, les outils de détection par anomalies génèrent un nombre important de faux positifs (i.e., des alertes d'attaques produites face à un comportement normal du système) et peuvent difficilement expliquer pourquoi un événement est considéré comme malveillant ; cette information est cruciale pour que l'analyste déclenche des contre-mesures. Récemment, les outils de détection hybrides se sont avérés prometteurs car ils présentent l'avantage d'être plus précis pendant la détection et adaptés pour le traitement de plusieurs sources d'événements (ex. paquets, logs). Toutefois, l'interopérabilité des divers formats d'événements pour le traitement est particuliè-

rement ardue avec ces outils. Par conséquent, les experts ont recours à l'écriture de plusieurs microprogrammes (scripts), l'usage de nombreux outils complémentaires et, éventuellement, la modification du code des outils de détection. Avec la croissance des attaques, ces opérations deviennent coûteuses et limitent considérablement la maintenabilité, la réutilisabilité et le réusinage des outils.

Objectifs

Le but de cette thèse est de proposer une approche formelle de détection d'intrusions par la corrélation d'événements. Une telle approche consiste à spécifier les comportements d'attaques dans un langage structuré et plus abstrait, indépendamment de tout logiciel ou matériel. De ce fait, les spécifications d'attaques sont facilement vérifiées, maintenues et réutilisées par les parties tierces d'une organisation. Nous considérons la méthode de spécification basée sur les diagrammes d'état-transition algébriques (ASTDs). Les ASTDs offrent une vue graphique, hiérarchique et modulaire des spécifications d'attaques. La corrélation d'événements est effectuée grâce à un interpréteur et un compilateur d'ASTDs que nous définissons et implémentons dans le cadre de cette thèse. Dans un but de détection, l'interpréteur permet d'exécuter les spécifications d'attaques sur différentes sources d'événements provenant du réseau ou de l'hôte. Tandis que le compilateur permet de générer un programme exécutable à partir d'une spécification ASTD de l'attaque. Le programme compilé va être exécuté pour la détection d'attaques. Due à l'hétérogénéité des événements, le langage proposé est couplé avec une ontologie pour la représentation des différents types d'événements avant leur envoi au détecteur.

Méthodologie

Le travail effectué durant cette thèse a débuté par une revue approfondie des domaines d'attaques et leurs vulnérabilités, des techniques d'attaques et de détection d'intrusions. Après cet aperçu contextuel, il était question de définir un langage de spécification de haut niveau en se basant sur les comportements observés des attaques et le langage existant ASTD. De ce fait, une simulation de plusieurs attaques a été

effectuée afin d'identifier les fonctionnalités à ajouter au langage ASTD. Ensuite, la prochaine étape a consisté à spécifier différentes attaques simples et complexes en utilisant la notation étendue. Afin de valider la notation, les spécifications d'attaques ont été présentées à des experts de la sécurité et une évaluation comparative a été réalisée entre le langage défini et d'autres langages de détection d'attaques existants. Pour l'évaluation, les performances de l'interpréteur du langage ont été comparées avec celles des outils industriels de détection d'attaques (ex. Snort, Zeek) à travers de nombreuses études de cas. Enfin, la dernière étape était de réaliser le compilateur du langage d'attaques. Pour ce faire, des règles de traduction ont été définies pour le passage du langage d'attaques au langage intermédiaire et du langage intermédiaire vers les langages de programmation de haut niveau tels que C++. Puis, le code généré a été optimisé et compilé en format binaire en utilisant les compilateurs natifs du langage de programmation (ex. GCC, JAVAC). Les performances des programmes générés sont comparées aux outils existants de corrélation d'événements (ex. iASTD, Beepbeep, MonPoly). La méthodologie ci-dessus se résume de la façon suivante :

1. Revue approfondie des domaines, techniques d'attaques et approches de détection d'intrusions (Chapitre 1).
2. Définition formelle et extension du langage ASTD existant pour la spécification d'attaques (Chapitre 2).
3. Méthodologie de spécification d'attaques et évaluation des performances de l'interpréteur du langage (Chapitre 3).
4. Définition formelle des règles de traduction du langage d'attaques au langage intermédiaire et du langage intermédiaire vers le langage C++ (Chapitre 4).
5. Comparaison des performances des programmes générés à partir du compilateur avec les outils de corrélation iASTD, Beepbeep, et MonPoly (Chapitre 4).

Contributions

Les contributions de cette thèse sont comme suit :

1. L'étude comparative et approfondie de plus de trois cent travaux récents sur les domaines d'attaques et leurs vulnérabilités (ex. contrôle industriel, médical), la

classification des systèmes de détection d'intrusions, les ensembles de données récents pour l'évaluation de tels systèmes, les techniques de détection d'intrusions et de traitement de flots d'événements hétérogènes, les limites et solutions potentielles pour la mitigation des attaques [311]. Cette étude a permis de sélectionner les approches les plus appropriées pour la conception et l'implémentation d'un système de détection d'intrusions. Ce travail a fait l'objet d'un article de revue publié dans le volume 4 de la revue *IEEE Communication Surveys and Tutorials* (IF : 29.830).

2. L'extension du langage ASTD et la définition formelle de sa sémantique. Cette extension a permis de prendre en compte la déclaration d'attributs (i.e., variables d'état), la déclaration des actions contenant du code exécutable pour modifier les attributs au cours de l'exécution d'une transition et dans l'état de l'automate (entrée, séjour, sortie), et un nouvel opérateur ASTD appelé *Flow* pouvant exécuter des événements partagés sur plusieurs modèles ASTDs lorsque cela est possible. Les actions peuvent être exécutées également au niveau d'un ASTD lui-même, afin de facilement factoriser le code à exécuter pour chaque transition de l'ASTD. La notation proposée a été implémentée dans deux interpréteurs, un en Prolog avec ProB, et l'autre en OCaml. ProB s'est avéré un ajout utile, car il donne accès à plusieurs fonctionnalités de contrôle de modèle déjà implémentées, telles que le contrôle du raffinement, le contrôle de terminaison et la vérification de formules temporelles [234]. Ce travail a fait l'objet d'un article publié dans le cadre de la $23^{ième}$ édition de la conférence internationale *ICECCS (International Conference on Engineering of Complex Computer Systems)* de rang A.

3. L'élaboration d'une méthodologie de spécification d'attaques avec la notation étendue des ASTDs et la comparaison de l'outil implémentant le langage avec des outils de détection d'attaques tels que Zeek et Snort. La notation a été couplée avec des descriptions de scénarios d'attaques tels que CAPEC [1] et ATT&CK [283]. L'évaluation des outils est effectuée en temps réel et sur des ensembles de données existants. Plus de 65 attaques ont été simulées et spécifiées avec la notation en collaboration avec Nokia Canada. Une vingtaine d'attaques a été spécifiée à partir d'énormes ensembles de données réelles provenant du Centre de la Sécurité des Télécommunications et de l'Université Technique Tchèque de Prague. Pour la corrélation d'événements, l'ou-

1. http ://makingsecuritymeasurable.mitre.org/docs/capec-intro-handout.pdf

til implémentant l'approche a produit peu de fausses alertes et est plus précis que ceux répertoriés dans la littérature [312]. Ce travail a fait l'objet d'un article publié dans le cadre de la $34^{ième}$ édition de la conférence internationale *AINA (International Conference on Advanced Information Networking and Applications)* de rang B.

4. La conception, l'implémentation, l'optimisation, et l'évaluation d'un compilateur du langage d'attaques vers le langage C++. Une syntaxe du langage intermédiaire a été définie ainsi que les règles traduction du langage d'attaques vers le langage intermédiaire et du langage intermédiaire vers un langage de programmation donné. Le code généré a été optimisé avec l'élimination des calculs redondants et le support de la Kappa optimisation pour l'exécution efficiente des opérateurs quantifiés. Les programmes compilés ont été comparés avec plusieurs outils de traitement d'événements (i.e., iASTD, Beepbeep, MonPoly) sur des ensembles de données de référence et sont plus rapides que ceux-ci.

Plan de la thèse

Le reste de ce manuscrit est organisé comme suit. Le Chapitre 1 fournit une revue et une évaluation des approches existantes sur la détection et le traitement d'événements. Ensuite, le Chapitre 2 décrit l'extension des ASTDs pour la détection d'attaques ainsi que sa sémantique formelle à travers une étude de cas. Le Chapitre 3 valide la notation proposée sur des données temps réel et de référence. La méthodologie de conception du compilateur du langage et la comparaison des performances des programmes compilés avec les outils de traitement d'événements existants (ex. iASTD) sont présentées dans le Chapitre 4. Les limites rencontrées durant les travaux de recherche, ainsi que les perspectives sont présentées en conclusion de cette thèse.

Il est à noter que l'annexe A illustre un cas d'attaque et décrit quelques règles de détection de l'attaque. L'annexe B présente des exemples de génération du code.

Chapitre 1

État de l'art

Résumé

De nos jours, les technologies réseaux sont essentielles pour transférer et stocker diverses informations sur les utilisateurs, les entreprises et les industries. Cependant, la croissance du taux de transfert d'informations étend la surface d'attaque, offrant un environnement riche aux intrus. Les systèmes de détection d'intrusion (IDS) sont des systèmes répandus capables de contrôler passivement ou activement les activités intrusives dans un hôte et un périmètre réseau définis. Récemment, différents IDS ont été proposés en intégrant diverses techniques de détection, génériques ou adaptées à un domaine spécifique et à la nature des attaques opérant sur ce domaine. Le paysage de la cybersécurité traite d'énormes flux d'événements diversifiés qui augmentent de manière exponentielle les vecteurs d'attaque. Les méthodes de traitement des flux d'événements (ESP) sont des solutions qui exploitent les flux d'événements afin de fournir des informations exploitables et une détection plus rapide. Dans cet article, nous décrivons brièvement les domaines ainsi que leurs vulnérabilités sur lesquels se fondent les articles ré-

cents. Nous étudions également les normes d'évaluation de la vulnérabilité et de classification des attaques. Ensuite, nous effectuons une classification des systèmes de détection d'intrusions, des métriques d'évaluation et des ensembles de données. Nous fournissons les détails techniques et une évaluation des travaux les plus récents sur les techniques IDS et les approches ESP couvrant différentes dimensions (axes) : domaines, architectures et technologies de communication locales. Enfin, nous discutons des défis et des stratégies pour améliorer l'IDS en termes de précision, de performance et de robustesse.

Commentaires

La contribution ici réside dans l'étude comparative et approfondie de plus de trois cent travaux récents sur les domaines d'attaques et leurs vulnérabilités (ex. contrôle industriel, transport, médical, construction, système sans pilote), la classification des systèmes de détection d'intrusions, les ensembles de données récents pour l'évaluation de tels systèmes, les techniques de détection d'intrusions et de traitement de flots d'évènements hétérogènes, les limites et solutions potentielles pour la mitigation des attaques [311]. Cette étude a permis de sélectionner les approches les plus appropriées pour la conception et l'implémentation d'un système de détection d'intrusions.

Les contributions décrites dans ce chapitre ont fait l'objet d'un article de revue publié dans le volume 4 de la revue *IEEE Communication Surveys and Tutorials*, le 12 juin 2019.

Les contributions et l'article sus-cité ont été élaborées par mes soins en tenant compte des remarques et commentaires issus de mon équipe d'encadrement.

Intrusion Detection Systems: A Cross-Domain Overview

Lionel N. Tidjon
Université de Sherbrooke, GRIF, Québec, Canada
Télécom SudParis, SAMOVAR, Institut Polytechnique Paris, France
lionel.nganyewou.tidjon@usherbrooke.ca
Marc Frappier
Université de Sherbrooke, GRIF, Québec, Canada
marc.frappier@usherbrooke.ca
Amel Mammar
Télécom SudParis, SAMOVAR, Institut Polytechnique Paris, France
amel.mammar@telecom-sudparis.eu

Keywords: Intrusion detection systems, vulnerabilities, vulnerability assessment, attack classification, intrusion detection techniques, event stream processing, datasets

Abstract

Nowadays, network technologies are essential for transferring and storing various information of users, companies, and industries. However, the growth of the information transfer rate expands the attack surface, offering a rich environment to intruders. Intrusion detection systems (IDSs) are widespread systems able to passively or actively control intrusive activities in a defined host and network perimeter. Recently, different IDSs have been proposed by integrating various detection techniques, generic or adapted to a specific domain and to the nature of attacks operating on. The cybersecurity landscape deals with tremendous diverse event streams that exponentially increase the attack vectors. Event stream processing (ESP) methods appear to be solutions that leverage event streams to provide actionable insights and faster detection. In this paper, we briefly describe domains (as well as their vulnerabilities) on which recent papers were based. We also survey standards for vulnerability assessment and attack classification. Afterwards,

we carry out a classification of intrusion detection systems, evaluation metrics and datasets. Next, we provide the technical details and an evaluation of the most recent work on IDS techniques and ESP approaches covering different dimensions (axes): domains, architectures, and local communication technologies. Finally, we discuss challenges and strategies to improve IDS in terms of accuracy, performance, and robustness.

1.1 Introduction

The global Internet economy is estimated to be valued at $4.2 trillion in 2016 [78]. It is estimated that the cost to the global economy from cybercrime is more than $400 billion [1]. Cyber-attacks are still spreading, targeting information systems, industrial infrastructures, computer networks, and personal end-devices. In most cases, vulnerabilities related to design flaws and human errors are the main causes of such attacks [11]. To defend network perimeters against these attacks, various intrusion prevention and detection systems have been proposed. Such systems are strategically placed in a network or host where they operate in single, distributed, centralized, collaborative or cooperative mode to prevent and/or detect eventual attacks against the confidentiality, integrity, and availability of information. With the growth of network infrastructures, intrusion detection systems must constantly adapt to new environments and changes. Network infrastructures support multiple diverse architectures such as Mobile Ad-hoc Networks (MANETs), Vehicular Ad-hoc Networks (VANETs), Unmanned Aerial Vehicle Networks (UAVNETs) and local communication technologies (e.g. Wi-Fi, Bluetooth) that are vulnerable to unknown/zero day attacks [207]. Consequently, network infrastructures require high performance and accurate IDSs.

An IDS is a hardware and/or software that passively or actively controls a network/host in order to find possible intrusions. Generally, IDSs are mainly divided into three categories: Network-based IDSs (NIDSs), Host-based IDSs (HIDSs), and Hybrid IDSs. A NIDS is located in a demilitarized zone (DMZ) just after the firewall. It captures network traffic in real time, analyzes it and performs defensive actions. These actions against malicious traffic can be either single alerts to the administrator or active reactions like dropping malicious traffic. Unlike NIDSs, HIDSs inspect log files of the operating system, services and softwares installed on the host and trigger alerts when suspicious activities are detected. IDSs that provide both NIDS and HIDS capabilities are called hybrid IDSs. A hybrid IDS uses both NIDS and HIDS for the gathering and analysis of heterogeneous cyber threat information. An IDS that can autonomously block attacks before they occur is called an Intrusion Prevention System (IPS).

1. www.mcafee.com/ca/resources/reports/rp-economic-impact-cybercrime2.pdf

1.1. Introduction

In recent years, a large number of research work has been published on the use of four main IDS techniques: i) anomaly-based detection that builds statistical patterns representing normal behaviours of the monitored system to identify abnormal patterns (e.g. machine learning, biologically-inspired systems); ii) knowledge-based detection also known as signature-based detection matches incoming signatures to those in its database of known threatening signatures for detection (e.g. knowledge-based systems); iii) multi-agent-based detection where IDS agents operate in centralized, distributed, collaborative and cooperative manner to detect advanced persistent threats and multi-stage attacks; and iv) hybrid detection that leverages multiple detection techniques to increase the detection accuracy (e.g. machine learning + knowledge-based systems + multi-agent systems). Recently, several detection techniques were focused on a particular application domain, although others were more generic. In addition, few recent papers were focused on IDS techniques in big network infrastructures particularly, event stream processing (ESP) approaches that are processor and resource-intensives.

Recent survey papers describe latest advances in a specific dimension (axis). Dimensions give an essential understanding of areas related to recent work, that will be classified and evaluated in our paper. This paper extends existing survey papers [59, 80, 153, 176, 208, 223, 230, 299, 350] by covering different comparison dimensions. The first dimension is the domain, a technological and general application area underlying a specific research work, for example: Cloud Computing, Big Data, Software Defined Networking, Smart Cities in public safety and Smart Grids in cyber-physical systems. The second is the architecture or topological organization of network objects, for example: MANET, VANET, and UAVNET. Finally, the Local Communication Technology (LCT) that enables exchanging data on a small and large distance, for example: Bluetooth, Zigbee, Wi-Fi and WiMAX. For a given dimension, we present some topics that are not fully covered in recent papers such as vulnerabilities, attack classification, intrusion detection techniques, event stream processing approaches, and dataset lists. In Table 1.1, we use two notations: ✓ when an aspect is covered by the survey and ✗ otherwise. Surveys [59, 350] cover the same domain, but with different topic coverage, indicated in Table 1.1 with two distinct marks. Dimensions are described in Section 1.2 including their vulnerabilities and attacks.

Table 1.1 – Comparison between recent surveys and ours

Dimension	[153, 223]	[80]	[59, 350]	[208]	[230]	[176]	[299]	ours
Domain	CloudC.	SCADA	BigData	SmartGrid	✗	✗	✗	✓
Architect.	✗	✗	✗	✗	MANET	✗	VANET	✓
LCT	✗	✗	✗	✗	✗	Wi-Fi	✗	✓
Topics								
Vulnerability	✓	✓	✗	✗	✗	✓	✓	✓
Attack class.	✓	✓	✗	✓	✗	✓	✓	✓
Detect. tech.	✓	✓	✗, ✓	✓	✓	✗	✓	✓
ESP methods	✗	✗	✓	✗	✗	✗	✗	✓
Dataset(list)	✗	✗	✓, ✗	✗	✗	✓	✗	✓

For the present survey, we have reviewed more than 250 research articles and selected about 100 qualitative articles published from December 2015 to January 2019 in top academic databases such as IEEE explore, Springer Link and Science Direct. We restricted ourselves to this period because we wanted to present the latest advances, precisely, new emerging aspects which are not yet covered in recent survey papers. The selection process was based on the dimension as mentioned above, the year of publication, the number of citations, and the publishing house.

The contributions of this paper include:

— A dimensional overview of environments where IDSs are used. It gives an essential understanding of domains and vulnerabilities related to the surveyed research papers.

— A presentation of standards for vulnerability assessment and attack classification;

— The technical details and an evaluation of recent work on intrusion detection techniques and event processing methods for each dimension using the following criteria: data source or input, technique, datasets, result or output;

— An up-to-date dataset list.

The rest of this paper is structured as follows. Section 1.2 describes dimensions and vulnerabilities of the surveyed papers, standards for vulnerability assessment and cyber attack classification. Section 1.3 gives an IDS classification including evaluation metrics and datasets. Section 1.4 reviews recent papers on intrusion detection techniques and evaluates them according to each dimension. In Section 1.5, we review

event stream processing methods and evaluate them according to each dimension using some criteria. Finally, Section 1.6 concludes and discusses the challenges of IDSs and strategies to improve IDSs in terms of accuracy, performance, and robustness.

1.2 Dimensions, vulnerabilities and attacks

Cyber-attack targets can be categorized in three dimensions: domain, architecture, and technology (see Fig. 1.1). Dimensions allow describing the vulnerable environments protected by IDSs and, in particular, understanding the security challenges in these environments. We also mention standards for assessing vulnerability and classifying cyber attacks.

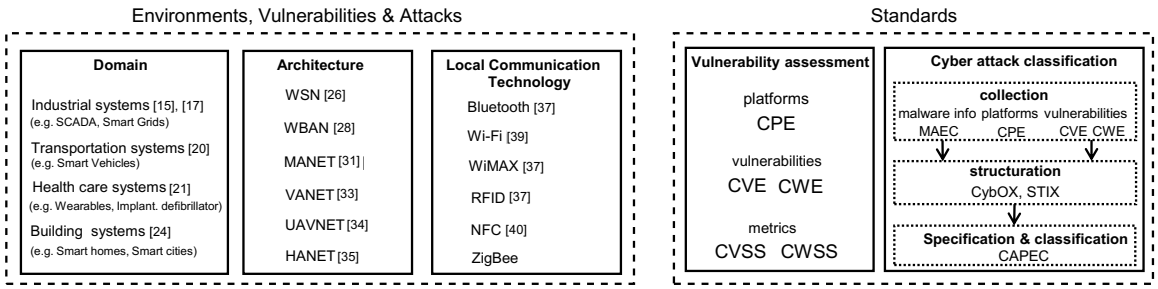


Figure 1.1 – Overview of IDS environments, vulnerabilities and standards

1.2.1 Domain

In the literature, intrusion detection has been addressed in many domains. The most important domains are defined below.

Industrial and control systems. Supervisory Control and Data Acquisition (SCADA) systems are extensively used in industries to automate the control of systems like power grids, nuclear facilities, water supply systems and enable the remote monitoring of industrial devices [240]. These devices are controlled using vulnerable LCTs like WiMAX, Wi-Fi or Bluetooth. LCTs allowed cyberattacks such as Night dragon and Stuxnet worm to penetrate SCADA networks and control industrial devices [316]. SCADA is used in Smart Grids to monitor relays, automated feeder switches and

batteries[2]. Smart grids are vulnerable to cyber attacks such as false data injection attacks (FDIAs), DoS (Denial of Service), and replay attacks [204].

Transportation systems. Intelligent transportation systems (ITS) is one solution that leverages modern advanced communication technologies to make transportation systems more efficient, reliable, secure, and resilient [206]. An ITS supports In-Vehicle, Vehicle-to-Vehicle, and Vehicle-to-Infrastructure communication (e.g. Bluetooth, Wi-Fi, WiMAX) that are integrated into embedded devices in smart vehicles, roads and railways. Recently, Symantec reported that numerous vehicle devices support vulnerable Linux-based operating systems[3] and wireless technologies that hackers can use to remotely take control, even to cut motor transmissions. ITS communications are also vulnerable to DoS/DDoS attacks, network protocol attacks, password and key attacks [256].

Health care systems. Ubiquitous health-care systems allow real-time patient care using biosensors (e.g. insulin pumps, wearables) to respond to individual requirements anywhere, anyhow, and at any time [305]. Recently, potential vulnerabilities were found in medical devices such as implantable cardiac defibrillators, CT-scanners and led to ransomware attacks [233]. In fact, health-care sensors support Wireless Body Area Network (WBAN) technologies in order to remotely access physiological data in real-time (e.g. ZigBee, Bluetooth, Wi-Fi). WBAN technologies are vulnerable to various attacks such as Man-In-The-Middle, Eavesdropping, DoS, that could be life-threatening [305].

Building systems. Building Systems integrate smart devices that continuously collect raw data, control building's heating, lighting systems, air conditioning and water heating using networked building automation systems [218]. Building systems are increasingly connected to the Internet, which increases the attack vectors, making the target environment vulnerable. Hence, Internet-facing building devices with insufficient authentication, default passwords, create opportunities for unauthorized external access [263].

2. https://energy.gov/oe/activities/technology-development/grid-modernization-and-smart-grid
3. https://www.symantec.com/content/dam/symantec/docs/reports/istr-21-2016-en.pdf

1.2.2 Architecture

Increasingly, network infrastructures support various dynamic and autonomous networking architectures in order to provide a high-availability, high-scalability, high-reliability, high-resiliency and low-latency of network connectivities in modern areas such as health care, transportation, industry and building. Some of these networking architectures are described hereafter.

Wireless Sensor Networks (WSN). WSNs are wireless networks of spatially distributed autonomous sensor nodes, with limited resources (e.g. battery energy), that monitor physical conditions (e.g. power usage, temperature), communicate together and transfer information for system management issues [184]. WSN sensors are frequently affected by network-layer vulnerabilities. They face both external and internal node attacks such as Sinkhole attacks, DoS attacks, Wormhole and Sibyl attacks [159].

Wireless Body Area Network (WBAN). A WBAN is a communication standard optimized for low power devices and operation on, in or around the human body (but not limited to humans), to serve a variety of applications including medical, consumer electronics, personal entertainment and others [323]. WBANs consist of autonomous or managed sensor nodes placed either in the body or on the skin to collect and monitor patients' health data (e.g. electro cardiogram, blood pressure) in real-time using vulnerable wireless technologies for end-uses. Health record data are extremely sensitive, particularly when an intruder can access, modify and inject false information that may result in a patient's death. WBANs are vulnerable to node capture attacks, impersonate and spoofing attacks [169].

Mobile Ad-hoc Network (MANET). A MANET is a self-configuring infrastructureless dynamic wireless networks where the network topology changes over time [215]. Mobile nodes, with resources restrained, autonomously adjust and communicate with each other. A MANET has numerous weak points including the absence of a fixed infrastructure, the dynamic topology change, their dependence on cooperative communication, the limited resource constraints and the unreliability of wireless links [226]. Consequently, they are vulnerable to various cyberattacks including gray-hole attacks, black-hole attacks, selective packet dropping attacks, Sybil attacks, and flooding attacks [302].

Vehicular Ad-hoc Network (VANET). A VANET is a part of MANET where vehicular nodes can move speedily inside the network, can communicate with each other or with roadside units (RSUs) at any intersection point and traffic light, using three communication modes: Vehicle-to-Vehicle (V2V), Vehicle-to-Infrastructure (V2I) and Hybrid [82]. A VANET is vulnerable to cyberattacks due to various constraints like the use of wireless channels necessary to exchange information (e.g. GPS, Wi-Fi), high mobility and dynamic topology [148]. VANET attacks are classified into three categories [148]: attacks on wireless interface, attacks on infrastructure, attacks on hardware and software.

Unmanned Aerial Vehicle Network (UAVNET). Unmanned Aerial Vehicles (UAVs) [116] are aircrafts which carry no human pilot or passengers. UAVs are recent technologies used in various areas such as agriculture for crop monitoring, medical for medical supply delivery, and military for border surveillance. UAVNETs are fluid infrastructure-based networks of airborne nodes (UAVs) [116]. Each UAV node can move slowly or rapidly and can communicate with ground nodes (i.e. control stations) as well as other UAVs. UANETs are vulnerable to jamming, injection fault and spoofing attacks.

Heterogeneous Ad-hoc Network (HANET). HANETs are autonomous, multi-hop, heterogeneous networks integrating both wireless sensor networks (WSNs), mobile ahhoc networks (MANETs), wireless body area networks (WBANs). They are accessible and interconnected through gateway nodes [257]. A HANET is vulnerable to WSN, MANET, and WBAN attacks.

1.2.3 Local Communication Technology

Local network technologies enable humans and things to transmit messages, video, files, and other data through peer-to-peer or/and client-server connections in various network architectures (e.g. MANET, VANET). They have various weak points according to the used technology such as short range communication, short authentication keys or passwords and vulnerable cypher algorithms.

Bluetooth. Bluetooth is a wireless technology standard for short-range radio communications, allowing transmitting data (e.g. audio, patient's record) at high speeds

using radio waves, between various devices (approximately 10 meters of each other) including computers, mobile phones, and other electronics. Bluetooth has many vulnerabilities such as authentication (no user authentication), keys (short size, shared master key), cryptography (weak cypher algorithm), and security services (no end-to-end security, no audit) [57]. Hence, Bluetooth is vulnerable to DoS, Man-In-the-Middle (MITM), Bluespoofing, Bluejacking, and Bluebugging attacks [38].

Wireless Fidelity (Wi-Fi). Wi-Fi is a Wireless Local Area Network (WLAN) based on IEEE 802.11 standards, that connects embedded devices within a short distance (less than 100 meters). Today, it is used to remotely control human health, SCADA systems, vehicles and unmanned aerial vehicles (e.g. drones) for public safety [131]. However, Wi-Fi is vulnerable to various cyberattacks targeting confidentiality (eavesdropping, Wep key cracking, MITM), integrity (frame injection, data replay) and availability (DoS, beacon flood, de-authenticate flood) [354].

Worldwide Interoperability for Microwave Access (WiMAX). WiMAX is a Wireless Metropolitan Area Network (WMAN) technology based on IEEE 802.16 standards, that provides broadband services for remote locations in different areas like public safety, medical, transportation and industry. As any other wireless technology, WiMAX introduces several security risks and it is vulnerable to cyberattacks at different layers: physical layer (jamming, scrambling) and MAC layer (rogue Base Station, MITM) [38].

Radio Frequency Identification (RFID) is a data collection technology that uses electromagnetic fields in the radio frequency (RF) portion of the electromagnetic spectrum to uniquely identify and track tags attached to objects. RFIDs are vulnerable to numerous attacks such as skimming, illicit tracking, unauthorized access and DoS [38].

Near Field Communication (NFC) is a wireless technology with a short range (about 20 centimeters) integrated into smart cards, smart phones or tablets, that allows wide services including identification, mobile payments, data exchange and other transactions. NFC are vulnerable to eavesdropping attacks, relay and jamming attacks [40].

ZigBee is a low-power consumption and low-cost wireless technology based on IEEE standard 802.15.4 and adapted for various embedded applications, medical

monitoring, industrial control and home automation. ZigBee may be vulnerable to tampering, key deciphering and authentication attacks.

1.2.4 Vulnerability assessment

The U.S. Department of Defense (DoD), Department of Homeland Security (DHS), and the Commerce Department's National Institute for Standards and Technology (NIST) collaborated and established vulnerability and cyber threat intelligence management standards through The MITRE Corporation [4]. The standards help cybersecurity companies to identify, characterize, structure, assess, report and share cyber attack information for prevention, detection and mitigations of cyber attack incidents. An incident occurs when a cyber attack exploits a vulnerability or weaknesses of a common platform (e.g. Windows 10) [328].

A vulnerability assessment is a process that defines, identifies, classifies and prioritizes vulnerabilities in a network, infrastructure and computer. It includes the following steps.

Defining network or system resources. The Common Platform Enumeration (CPE) is a standard that allows defining vulnerable IT products and platforms by providing a machine-readable format for encoding names.

Identifying and classifying vulnerabilities. The Common Vulnerability Enumeration (CVE) and Common Weaknesses Enumeration (CWE) gives a standard dictionary and classification for publicly known cyber security vulnerabilities and weaknesses respectively.

Prioritizing or assigning relative levels of importance to vulnerabilities. Vulnerabilities and Weakness levels are evaluated according to various metrics such as impact, environmental, temporal, attack surface and exploitability. The Common Vulnerability Scoring System (CVSS) and Common Weaknesses Scoring System (CWSS) are standards that provide consistent and flexible mechanisms, to prioritize vulnerability and weakness risks respectively.

4. https://www.mitre.org/capabilities/cybersecurity/overview/cybersecurity-resources/standards

1.2.5 Cyber Attack classification

MITRE's Adversarial Tactics, Techniques, and Common Knowledge (ATT&CK) describes cyber attacks' life cycle in 7 steps: Reconnaissance, Weaponization, Delivery, Exploitation, Control, Execution and Maintain (see Fig. 1.2). During the attack process, an attacker usually looks for unknown vulnerabilities on the target (Reconnaissance), creates remote access malware weapon (e.g. worm, virus) using vulnerabilities found (Weaponization), transmits the malware weapon (Delivery), uses the weapon to command and control remotely the victim (Exploit, Control) and executes malicious commands in the target machine for data ex-filtration and data destruction (Execution, Maintain).

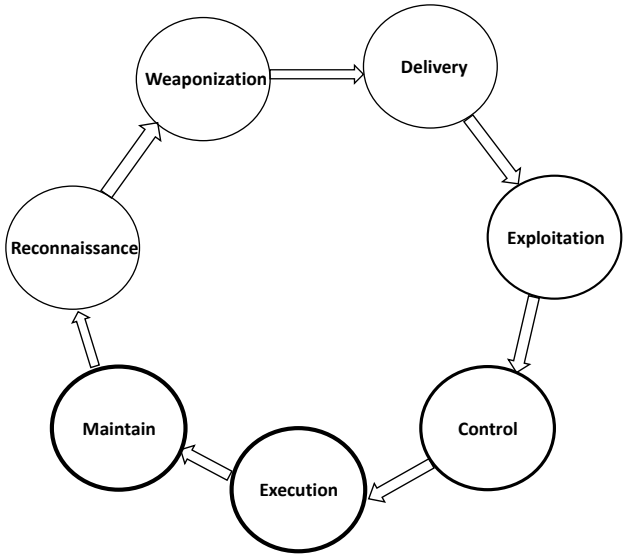


Figure 1.2 – Attack cycle

Cyber-attack classification consists in describing and categorizing attacks using several criteria such as mechanisms, domains, scenarios, impacts, consequences or effects of attack, motivations, skills and knowledge of attacker. Generally, the classification process is divided into two main steps:

Gathering and structuring cyber attack information. Standard platform, weakness, vulnerability and malware information are collected by MITRE's Common Platform Enumeration (CPE), Common Weaknesses Enumeration (CWE), Common Vulnera-

bility Enumeration (CVE) and Malware Attribute Enumeration and Characterization (MAEC) [5]. The resulting cyber attack information are represented in CybOX (Cyber Observable eXpression) language, and structured in STIX (Structured Threat Information eXpression) language for classification, analysis, storing and sharing.

Classifying cyber attacks. Once the attack is identified, it is specified and classified using the Common Attack Pattern Enumeration and Classification (CAPEC). CAPEC is a public dictionary and classification taxonomy of common attack patterns described in a structured and comprehensive XML schema. Attack patterns describe attack scenarios (i.e. execution flows) in term of phases and steps, attack impact, attack techniques and motivations.

1.3 IDS classification, metrics and datasets

IDS accuracy and performance depend on various properties such as product, type, structure, detection technique, state awareness, usage frequency, environment and response type.

1.3.1 IDS-centered Classification

In the literature, several types of IDS have been proposed. These systems can be classified according to different factors described below (see Fig.1.3).

Product. When deciding on IDS products, IT companies must balance or conduct a trade-off analysis between cost, performance, essential functionalities and risks. On the market, IDS products can be divided into three categories, depending on the vendor [128]: commercial-off-the-shelf (COTS) IDS, modifiable-off-the-shelf (MOTS) IDS and government-of-the-shelf (GOTS) IDS. A COTS IDS is a commercial IDS product that can be bought in the marketplace and fully designed for easy installation and interoperability (e.g. IBM QRadar, LogRhythm, Splunk). While a MOTS IDS is a modifiable COTS IDS product that can be customized by a third party to meet customer needs (e.g. Snort). Finally, GOTS IDSs are products developed by govern-

5. https://www.mitre.org/capabilities/cybersecurity/overview/cybersecurity-resources/standards

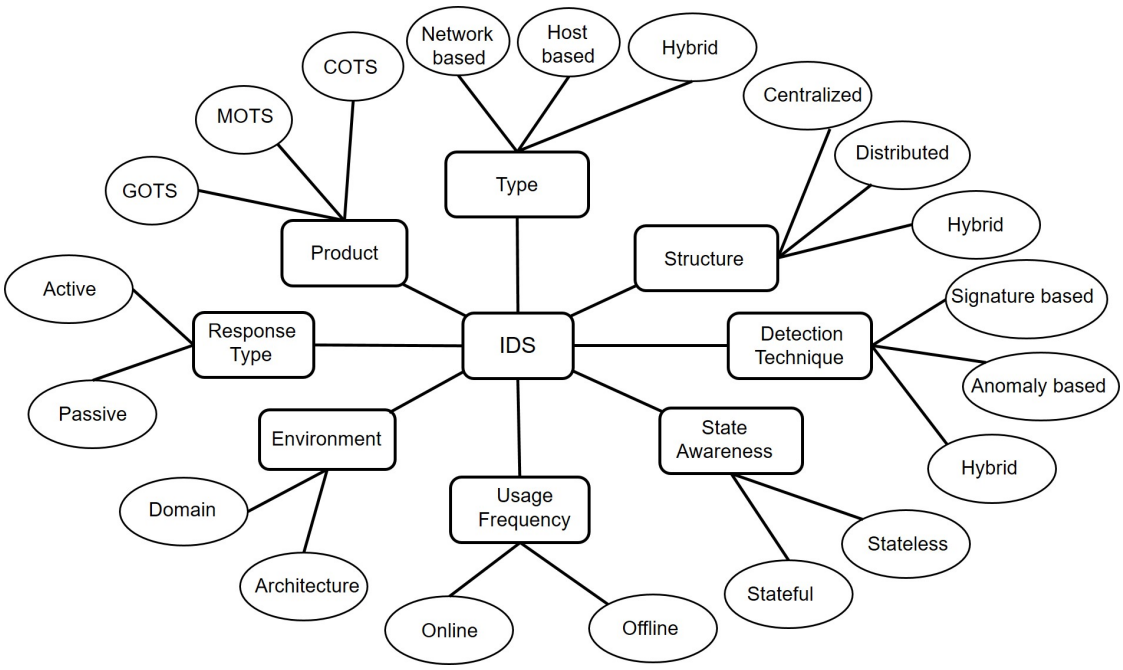


Figure 1.3 – An IDS-centered classification diagram

ment technical staff or a government-funded external company to meet government requirements and objectives (e.g. CSE AssemblyLine, US-CERT Einstein).

Type. Most IDSs are categorized in three classes: Network-based IDS, Host-based IDS and Hybrid IDS (see Section 1.1).

Structure. In a network topology, there are three main IDS classes: centralized IDS, distributed IDS and hybrid IDS (from a structure point of view). In centralized IDS, agents analyze network nodes or hosts and generate heterogeneous alerts, which are sent to a central C&C handler, responsible for aggregating, correlating, and making an accurate decision. In a distributed IDS, each agent is responsible of its node where it analyzes events and triggers alerts. There are two kinds of distributed IDS: Collaborative Distributed IDS (CDIDS) and Individual Distributed IDS also known as standalone IDS. A CDIDS is a system of IDS agents which collaborate or exchange information (P2P) together, without necessarily an administrator to detect distributed attacks such as DDoS and worm attacks [205]. A standalone IDS is limited to the monitored node and cannot take advantage of the other nodes to make accurate decisions. A hybrid IDS combines both centralized and distributed IDS in

big network topologies.

Detection technique. There are three IDS classes: Knowledge-based IDS also known as Signature or Misuse, Anomaly-based IDS also known as Profile or Heuristic, and Hybrid IDS. A knowledge-based IDS matches attack patterns or signatures to those in its signature database and triggers alerts whether a match is found. It cannot detect zero-day attacks but it can accurately detect known attacks. An anomaly-based IDS builds a normal behaviour profile of the monitored system and any deviation from the profile is considered as an abnormal activity. An anomaly-based IDS can detect zero-day attacks but generates numerous false positives. Currently, most companies combine both signature-based and anomaly-based IDSs, known as hybrid IDSs, to increase accuracy.

State awareness. IDSs can be divided in two categories: stateful and stateless IDS. A stateful IDS analyzes events and relates them to past events, usually kept in a state memory for that purpose. A stateless IDS analyzes an event without taking into account other events. A stateless IDS is less accurate in detecting Advanced Persistent Threat (APT) or multi-step attacks than a stateful IDS.

Usage frequency. In term of usage frequency, IDSs can be classify into two types: offline and online IDS. An offline or *a posteriori* IDS analyzes events stored in a database and uses contextual information to detect abnormal behaviors. While, an online or *on the fly* IDS analyzes events in real-time as they occur.

Environment. Each environment requires an IDS adapted to its constraints. In Cloud computing, Cloud IDSs can be deployed on virtual networks, hypervisor and virtual hosts that have limited resources (e.g. CPU, Memory), often defined by the provider. In MANET, MANET IDSs should be optimized for the dynamic network topology and highly constrained nodes. In big data infrastructures, big data IDSs analyze big heterogeneous event sources and should be accurate.

Response type. An IDS can also be classified as active or passive, depending on the action taken when a malicious activity is detected. Passive IDSs log malicious activities and trigger notification messages, which may be later consulted by system administrators. Active IDSs can take immediate action themselves, for instance reject malicious traffic, close a port.

Table 1.2 – Validation technique used

Year	Dataset	Provider	Source	Attacks	IDS
1998	DARPA 1998	Defense Advanced Research Projects Agency	Traffic+Audit data	DOS, Probe, R2L, U2R, Data attacks	NIDS, HIDS
1999	KDD cup 1999	MIT	Traffic	DOS, Probe, R2L, U2R	NIDS
1999-2000	DARPA 1999 DARPA 2000	Defense Advanced Research Projects Agency	Traffic+Audit data	DOS, Probe, R2L, U2R, Data attacks	NIDS, HIDS
2006	UNM	University of New Mexico	System call traces	Buffer overflows, symbolic link attacks, and Trojan programs	HIDS
2008	CAIDA Backscatter	Center for Applied Internet Data Analysis	Traffic	DOS attacks	NIDS
2009	ASNM CDX	Brno University Security	Traffic	Buffer overflow attacks	NIDS
2010	HTTP CSIC	Spanish National Research Council	Traffic	Static attacks, dynamic atttacks, unintentional illegal requests	NIDS
2010	ISOT	University of Victoria	Traffic	Storm and Waledac botnets	NIDS
2010-2011	MalGenome	Android Malware Genome Project	Malware signatures	Malwares (49 families)	NIDS, HIDS
2011	CTU	Czech technical university	Traffic	Click Fraud, Port scan, Botnet, DDoS, SPAM, IRC	NIDS
2011	MAWI (MAWILab)	MAWI Working Group	Traffic	Sasser worm, ping, netbios attacks, rpc and smb attacks	NIDS
2011	USMA CDX	National Security Agency / United States Military Academy	Traffic + Logs (Snort, DNS, Web Servers, Splunk)	--	NIDS, HIDS
2012	MACCDC	Mid-Atlantic Collegiate Cyber Defense Competition	Traffic + Audit logs (Bro,Snort and ICS)	APT, Malwares, Exploit, SSH login attempts	NIDS, HIDS
2012	SMS Spam Collection	University of California, Irvine	SMS messages	Spam	NIDS
2012	TUIDS	Tezpur University	Traffic	DDoS	NIDS
2010-2012	Drebin	Technische Universitat Braunschweig	Malware signatures	Android Malware (179 families)	NIDS, HIDS
2013	Gas Pipeline	Mississippi State University and Oak Ridge National Laboratory	No Events, Natural and Attack Events	Response injection, Dos, Command Injection	NIDS
2009-2013	WITS	Waikato Internet Traffic Storage	Traffic	--	NIDS
2013-2014	ADFA-LD/WD	University of New South Wales	Process traces	Hydra, Shell, Exploits, Backdoors	HIDS
2014	Gas Pipeline and Water Storage Tank	Mississippi State University	No Events, Natural and Attack Events	Recon, Dos, Code Injection, Command Injection, Reponse Injection, Fault replay, relay	NIDS
2014	Power System Attack	Mississippi State University and Oak Ridge National Laboratory	No Events, Natural and Attack Events	Short-circuit fault, Data Injection, Relay setting change, Remote tripping command injection, Line maintenance	NIDS
2015	UNSW-NB15	Australian Centre for Cyber Security (ACCS)	Traffic + Audit traces	Backdoors, Dos, Exploits, Generic, Reconnaissance, Shellcode and Worms	NIDS, HIDS
2015	Microsoft Malware Classification Challenge (BIG 2015)	Microsoft	Malware signatures	Malwares (9 families)	NIDS, HIDS
2006-2015	KYOTO	Kyoto University	Traffic	--	NIDS
2010-2016	AMD	Argus Cybersecurity Lab-University of South Florida	Malware signatures	Malwares (71 families)	NIDS, HIDS
2017	DDoSTB	I.K.G. Punjab Technical University	Traffic	DDoS	NIDS
2018	Contagio Malware Dump	International Cyber Threat Task Force	Malware signatures	APT, Malwares, Crime or Metasploit	NIDS, HIDS
1996-current	Defcon Capture the Flag (CTF)	United States Armed Forces - CTF competition	Traffic	Port scan, Port sweeps, buffer overflows, gain privilege, FTP attacks	NIDS
2002-current	USC/ISi ANT	University of Southern California	Traffic	DDoS, port scans, or worm outbreaks	NIDS
2005-current	CRAWDAD Wireless and Bluetooth	Community Resource for Archiving Wireless Data/Dartmouth College	Traffic	--	NIDS
2009-current	UNB ISCX (NSL-KDD, IDS,Android)	University of New Brunswick	Traffic, malware signatures	DOS, Probe, R2L, U2R, Malwares, Tor, Botnets	NIDS, HIDS

1.3.2 Metrics

In the literature, several metrics have been used in recent work to evaluate IDSs. These metrics measure their accuracy and performance. Table 1.3 provides a summary of these metrics.

Table 1.3 – Detection system - Evaluation Metrics

Sensivity/Recall/True Positive Rate/Detection Rate (DR) probability that the IDS outputs an alert when there is an intrusion	$\mathbf{DR}(\%) = \frac{\mathbf{TP}}{\mathbf{P}} = \frac{\mathbf{TP}}{\mathbf{TP+FN}}$
Fall-out/False Positive Rate (FPR) probability that the IDS outputs an alert although the behaviour of the system is normal	$\mathbf{FPR}(\%) = \frac{\mathbf{FP}}{\mathbf{N}} = \frac{\mathbf{FP}}{\mathbf{FP+TN}}$
Miss Rate/False Negative Rate (FNR) probability that the IDS does not output an alert although the behaviour of the system is malicious	$\mathbf{FNR}(\%) = \frac{\mathbf{FN}}{\mathbf{P}} = \frac{\mathbf{FN}}{\mathbf{FN+TP}}$
Specificity/True Negative Rate (TNR) probability that the IDS outputs no alter when the behaviour of the system is not malicious	$\mathbf{TNR}(\%) = \frac{\mathbf{TN}}{\mathbf{N}} = \frac{\mathbf{TN}}{\mathbf{TN+FP}}$
Bayesian Detection Rate/Positive Predictive Value/Precision probability that there is an intrusion when the IDS outputs an alert	$\mathbf{Precision}(\%) = \frac{\mathbf{TP}}{\mathbf{TP+FP}}$
Overall Sucess Rate/Accuracy (ACC) probability that the IDS outputs correctly when the behaviour of the system is normal and malicious	$\mathbf{ACC}(\%) = \frac{\mathbf{TP+TN}}{\mathbf{P+N}} = \frac{\mathbf{TP+TN}}{\mathbf{FP+TP+FN+TN}}$
F-measure/F-score harmonic mean of Precision and DR	$\mathbf{F\text{-}Score}(\%) = \frac{2\times\mathbf{DR}\times\mathbf{Precision}}{\mathbf{DR+Precision}}$
Negative Predicitive Value (NPV) probability that there is no intrusion when the IDS does not output an alert.	$\mathbf{NPV}(\%) = \frac{\mathbf{TN}}{\mathbf{TN+FN}}$
Receiver Operating Characteristic (ROC) curve shows the ability of the intrusion classifier	$\mathbf{TPR = f(FPR)}$

Let us consider four well-known parameters [92]: False Positive (FP) is the num-

ber of normal events misclassified as malicious, False Negative (FN) is the number of malicious events misclassified as normal, True Positive (TP) is the number of malicious events correctly classified as malicious and True Negative (TN) is the number of normal events correctly classified as normal. The number of normal events is denoted by N and the number of malicious events is denoted by P.

1.3.3 Datasets

As cyber attacks evolve, the cybersecurity landscape changes and the older datasets, usually considered in experiments, may not be relevant for recent security challenges. Nonetheless, well-known DARPA (Defense Advanced Research Projects Agency) and KDD cup (Knowledge Discovery in Databases) datasets are still widely used today [350]. KDD cup 1999 is a data set used for The Third International Knowledge Discovery and Data Mining Tools Competition, where the task was to build a predictive model able to classify normal connections and malicious connections. It consists of about 5 millions connection records that have 41 features for training and 24 attacks divided in 4 categories [308]: Probe (information gathering), User to Root (attempting to access root privilege from an user privilege), Remote to Local (trying to gain access to an unauthorized network/host from a remote machine) and DoS (making resources unavailable for legitimate users). Since, KDD cup 1999 was criticized about the characteristics of the synthetic data [308]. Firstly, synthetic data is not representative of existing real networks (network landscape changes). Secondly, it is not an exact definition of the attacks (attack behaviours evolve) and redundant records.

Other datasets have been proposed to overcome the KDD dataset problems. The first one is NSL-KDD provided by the Information Security Centre of Excellence at the University of New Brunswick. It is very similar to KDD cup 1999 with new advantages [308]: no redundant records i.e. learning models are not biased by other models which have better detection rates, the number of selected records from each difficulty-level group is inversely proportional to the percentage of records in the original KDD cup i.e. classification rates of distinct learning methods vary in a wider range, and the number of records is reasonable i.e. easy to run experiments on the

entire set without randomly selecting a small portion. However, NSL-KDD is not a perfect representation of existing real networks [308].

Real NIDS data sets are private because of confidentiality problems, i.e., they contain sensitive and private data that cannot be shared in the public domain (e.g., user information, company information, health records). There exist numerous other NIDS datasets [134] such as Kyoto, CAIDA, DEFCON, WAIKATO [107,133], HTTP CSIC 2010, TUIDS [25,26], DDoSTB [37,39], and USC Ant provided by academic and governmental institutions. In addition, we have found some HIDS datasets provided by the University of New Brunswick (UNB ISCX Android), the University of New Mexico (UNM datasets/s-tide), the University of New South Wales (ADFA-LD/WD), the United States Military Academy (USMA CDX), the Australian Centre for Cyber Security (UNSW-NB15), Microsoft Malware Classification Challenge, and Contagio Malware Dump. They offer rich system call traces of various active processes (malicious, normal) and malware signatures to evaluate HIDSs. We have summarized, in Table 1.2, well-known NIDS and HIDS datasets for evaluation.

1.4 Review and Evaluation of IDS techniques

Improving the accuracy and performance of IDSs remains an overwhelming problem. To tackle these challenges, multiple IDS approaches have been proposed in the surveyed papers. Anomaly-based detection takes advantage of recent AI techniques like Machine Learning (ML), biologically-inspired systems to accurately identify threatening behavioral patterns. Knowledge-based detection leverages expert systems, knowledge base systems, multi-agent systems to find meaningful suspicious signatures. Hybrid detection is increasingly used and seems getting better results by combining approaches above in different ways. Hereafter, we structure and present a dimensional overview of recent work on the aforementioned detection techniques (see Fig. 1.4). Tab. 1.4 shows the summary of papers in different dimensions (domain, architecture, technology) using criteria such as data source or input, technique, datasets, and output. We also use the notation DR (D) which denotes the intrusion detection rate, ACC (A) the accuracy rate, FPR (F) the false positive rate and ROC the Receiver Operating Characteristic curve area [92]. The ROC area is an integral

number of the function f that takes a FPR in parameter and returns a TPR.

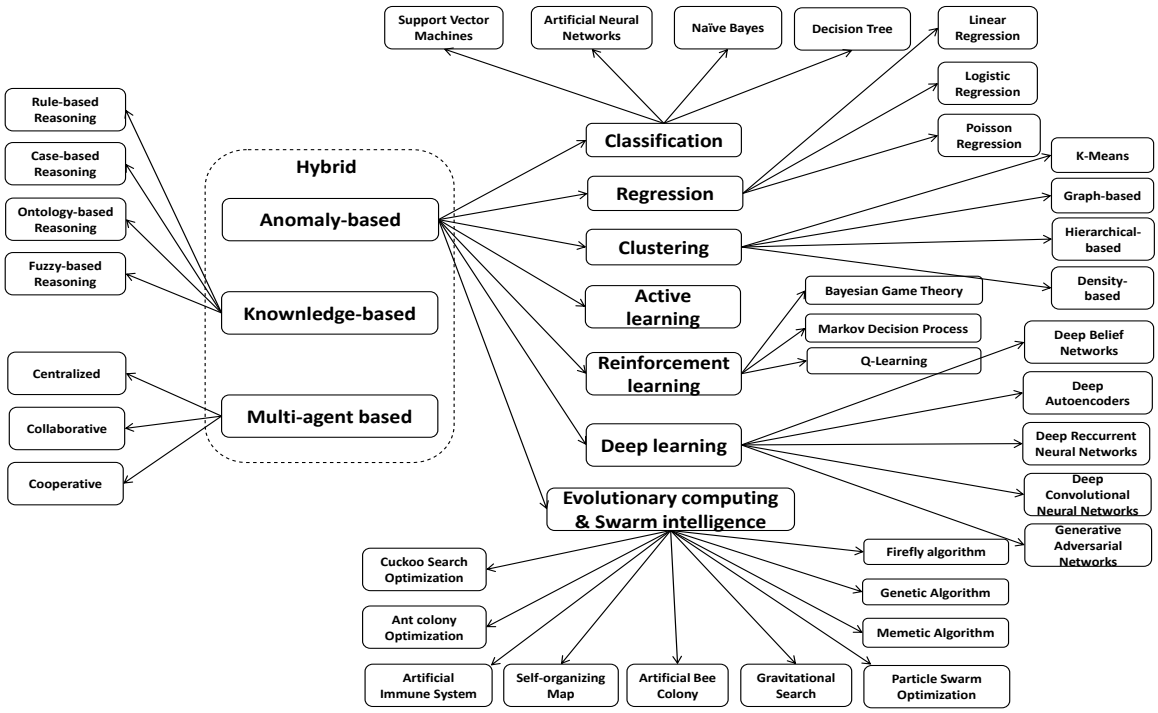


Figure 1.4 – Some recent detection techniques

1.4.1 Anomaly-based detection techniques

Anomaly-based detection is a well-known technique, used to find out behavioral patterns that do not match the normal behaviour profile of the monitored system. It can be formulated into four essential aspects [55]. The first one is the *nature of input data*, defined using a set of categorical, binary or continuous data instance's attributes. Another aspect is the *type*. Anomalies can be individual (one data instance), contextual (depending on the context) and collective (multiple data instances). The last aspect is the *labeling of data*. It consists in labeling data instances as normal or abnormal, using anomaly-based detection approaches. In general, we have three main approaches described in the following.

(a) supervised learning. Supervised learning is a machine learning approach that uses a training (known) dataset of inputs and their output values to make predictions

by trying to learn a function that correctly predicts the output values for a new data set. Two well-known examples of this approach are classification and regression. Supervised learning is a widely used technique but it has many drawbacks [185]. It needs known input and output datasets to train, which is not easily applicable in real-world systems, performs well on training datasets but poorly with new dataset (overfitting), and requires a lot of computation time for training.

(b) unsupervised learning. Unsupervised learning [23] is a machine learning approach that uses datasets consisting of input data without labeled responses. The well-known unsupervised learning method is clustering, which targets homogeneous data clusters such that the most similar data (aka low similarity distance) belongs to the same cluster. Inversely, heterogeneous data are separated in different clusters. The clusters are modeled using a measure of similarity which is defined upon metrics such as Euclidean, Manhattan, Tchebychev, and probabilistic distance.

(c) semi-supervised learning [69] assumes the desired output values are provided only for a subset of the training data and the remaining data is unlabeled. Semi-supervised learning is more widely applicable than supervised techniques. Recent hybrid intrusion detection approaches are semi-supervised by integrating methods like classification and clustering.

The last aspect is the *output* of anomaly detection and it contains two elements: *labels* to assign a label (normal, abnormal) for each test instance and *scores*, which is an anomaly degree. The following sub-sections describe the techniques of the main approaches mentioned above and recent work using these techniques.

1.4.1.1 Classification techniques

Classification [237] is a supervised approach, applied for categorical output values, where data can be separated into specific classes (see Fig. 1.5).

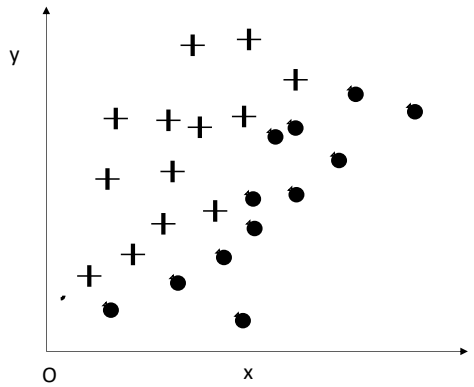


Figure 1.5 – Classification: An Overview

Theory. Let $\mathcal{D} = \{(x_1, y_1), ..., (x_n, y_n)\}$ be a training set of examples, where each $x_i \in \mathbb{R}^d$ (data) and $y_i \in \{-1, 1\}$ (2 classes), $i \in \{1, ..., n\}$. The prediction function is of the form [36],

$$y(x, w) = w^T \phi(x) + b$$

where $w = (w_1, ..., w_{n-1})^T$, $\phi = (\phi_1, ..., \phi_{n-1})^T$ is a basis function that can be linear or non-linear, and b is the bias weight; it has value w_0. There are many choices for the basis functions. For polynomial regression, $\phi_i(x) = x^i$. Gaussian basis functions are of the form,

$$\phi_i(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{1}{2}\left(\frac{x - \mu_i}{\sigma}\right)^2}$$

where μ_i governs the location of the basis functions in input space and σ their spacial scale. The optimization problem is to find an optimal w that minimizes the error $E_{\mathcal{D}}(w)$,

$$w = \arg\min_{w} \sum_{i=1}^{n} l(y(x_i, w), y_i)$$

1.4. Review and Evaluation of IDS techniques

Table 1.4 – Evaluation of recent intrusion detection techniques

Papers	Domain	Architect.	LCT	Source	Technique	Parent Tech.	Datasets	Results
M. Al-Rubaie *et al.* [4]	CloudC.	✗	✗	Traffic	SVM	anomaly	Simulation	Good
S. Doss *et al.* [86]	✗	MANET	✗	Traffic	SVM	anomaly	MITLincoln	Good
A. S. Sadiq *et al.* [280]	✗	MANET	✗	Traffic	ANN+PSO	hybrid	KDDcup99	D:99.5%
M. Barni *et al.* [31]	Health sys.	✗	✗	Traf.+ECGs	ANN+DT	hybrid	Simulation	Good
H. H. Pajouh *et al.* [137]	IoT	✗	✗	Traffic	NaiveB+kNN	hybrid	KDDcup99	D:84.8%; F:5.2%
U. Adhikari *et al.* [14]	SmartGrids	✗	✗	Traf.+Elec.Power	HA-DT	hybrid	Simulation	D:>94%
H. J. Patel *et al.* [255]	✗	✗	Zigbee	Fingerprint	RNDF+MCA	hybrid	Simulation	D:90%
K. Bu *et al.* [53]	✗	✗	RFID	RF signal	BF-Tree	anomaly	Simulation	Good
J. Zhang *et al.* [347]	SmartGrids	✗	✗	Elec.Power	MLR	hybrid	IEEE bus RTS	Good
K. Peng *et al.* [249]	BigData	✗	✗	Traffic	K-means+PCA	hybrid	KDDcup99	Good
C. Sudipta *et al.* [62]	Big Data	✗	✗	Traffic	GBC	anomaly	CTU-13	Good
K. Rina *et al.* [269]	✗	WSN	✗	RF signal	Hierar.Clust.	hybrid	KDDcup99	D:84.84%; 5.21%
C. Zhang *et al.* [352]	Network	✗	✗	Traffic	Density+SVDD	hybrid	KDDcup+UCI	D:95.2%
M. H. Eiza *et al.* [143]	✗	VANET	✗	Gen.Traffic	ACO	anomaly	Simulation	Good
M. Korczynski *et al.* [170]	✗	WSN	✗	Traffic	ABC	anomaly	Testbed	A:94.51%
L. Yang *et al.* [341]	SmartGrids	✗	✗	Elec.Power	ABC+ELM	hybrid	IEEE bus	D:86.8%; 11.06%
D. Tirtharaj *et al.* [73]	✗	WSN	✗	Traffic	GSA+PSO+ANN	hybrid	NSL-KDD	D:98.13%
C. Chen *et al.* [65]	✗	WSN	✗	Net.Sensors	MA+HRA	hybrid	Simulation	Good
M. H. Ali *et al.* [2]	Network	✗	✗	Traffic	PSO+DPFNN	hybrid	KDDcup99	A:99%
R. Roman *et al.* [273]	IoT	✗	✗	Net.Sensors	AIS	anomaly	Simulation	Good
J. M. Vidal *et al.* [322]	✗	WSN	✗	Net.Sensors	AIS	hybrid	KDD+CAIDA	Good
L. Xiao *et al.* [334]	✗	UAVNET	✗	Traffic	Q-Learning	RL	Simulation	Good
L. F. Maimo *et al.* [104]	✗	MANET	✗	Traffic	DBN+RNN+SAE	hybrid	CTU	D:70.95%
S. Garg *et al.* [117]	SDN	✗	✗	Traffic	RBM+SVM	hybrid	CMU	Good
N. Moustapha *et al.* [213]	CloudC.	✗	✗	Traffic	GAN+ODM	hybrid	KDD+UNSW	Good
R. Mitchell *et al.* [211]	Health sys.	✗	✗	Traf.+Health.Rec.	RBR	Knowledge	Simulation	D:92.4%; F:0.66%
D. Malathi *et al.* [87]	Health sys.	✗	✗	Health.Rec.	CBR+kNN	hybrid	UCI	Good
W. Li *et al.* [196]	Health sys.	✗	✗	Health.Rec.	FBR	hybrid	Simulation	Good
N. Naik *et al.* [232]	Health sys.	✗	✗	Traffic	FBR+GA	hybrid	UCI	Good
T.R.B. Kushal *et al.* [177]	SmartGrids	✗	✗	Elec.Power	MAS	Hybrid	Simulation	Good
T. Kim *et al.* [175]	Network	✗	✗	Malw. signatures	MultiDNN	hybrid	Malgenome	A:98%
M. Al-Qatf *et al.* [13]	Network	✗	✗	Traffic	SAE+SVM	hybrid	NSL-KDD	A:84.96%
I. H. A. *et al.* [19]	SDN	✗	✗	Traffic	CS+EGA	hybrid	Simulation	Good
M. Watson *et al.* [330]	CloudC.	✗	✗	Traffic	SVM	Anomaly	Testbed	A:>90%
K. Huang *et al.* [152]	✗	WSN	✗	Gen. Traffic	SOM	Anomaly	Simulation	D:95.3%; F:4.1%
H. Sedjelmaci *et al.* [300]	✗	UAVNET	✗	Gen. Data	Bayes. Game	Anomaly	Simulation	A:96-98%
P. Jockar *et al.* [158]	Smart Grids	✗	ZigBee	Traffic	Q-Learning	Anomaly	Testbed	D:92.5%; F:0%
J. Kevric *et al.* [172]	Network	✗	✗	Traffic	random+NBTree	Hybrid	NSL-KDD	A:89.24%
B. Hamid et al. [50]	Network	✗	✗	Traffic	GS+MI	Hybrid	NSL-KDD	D:88.36%; F:8.88%
D. P. *et al.* [251]	Network	✗	✗	Traffic	GA+DT	Hybrid	KDDcup99	A:98.5%; F:0.75%
M. G. *et al.* [275]	Network	✗	✗	Traffic	GA	Anomaly	NSL-KDD	D:95.32%; F:0.83%
Z. Yi *et al.* [342]	Network	✗	✗	Malw. samples	PSO	Anomaly	VirusTotal	A:>98%
M. E. A. *et al.* [5]	–	✗	Wi-Fi	Traffic	DAE+ANN	Hybrid	AWID	A:99.91%; F:0.012%
S. Shah *et al.* [292]	Network	✗	✗	Gen. Traffic	SVM+Firefly	Hybrid	Simulation	A:95%; F:8.6%
N. Nissim *et al.* [231]	Network	✗	✗	Ms Docs	active learn.	Anomaly	Testbed	D:94.4%; F:0.19%
G. Xu *et al.* [333]	Network	✗	✗	–	OBR	Knowledge	–	Good
Y. Chuan *et al.* [64]	Network	✗	✗	Traffic	RNN	Anomaly	NSL-KDD	A:99.81%
R. Kwon *et al.* [174]	Network	✗	✗	Traff+Audit Logs	Reg.+Corr.	Hybrid	Keimyung U.	Good
K. Grosse *et al.* [120]	Network	✗	✗	Malw. Samples	GAN	Anomaly	DREBIN	A:>95%
M. S. Parwez *et al.* [253]	Big Data	✗	✗	Traffic	K-means+Hierar.	Anomaly	Testbed	Good
K. Wang *et al.* [327]	Smart Grids	✗	✗	Traf.+Elec. Power	Bayes. Game	Anomaly	Test-bed	Good

C. Feng *et al.* [101]	Smart Grids	✗	✗	chemical samples	GAN	Anomaly	SWaT	Good
W. Laftah *et al.* [18]	Network	✗	✗	Traffic	SVM+ELM	Hybrid	KDDcup99	A:95.75%
S. Roshan *et al.* [266]	Network	✗	✗	Traffic	Clustering+ELM	Hybrid	NSL-KDD	D:84%;F:3%
M. Chen *et al.* [58]	Network	✗	✗	Traffic	AIS+PBIL	Hybrid	KDDcup99	A:97.03%
R. Aamir *et al.* [17]	Network	✗	✗	Traffic	Fuzzy	Anomaly	NSL-KDD	A:84.12%
F. Y. Leu *et al.* [202]	Network	✗	✗	Traffic	DM	Anomaly	Testbed	D:94.29%
S. Chin Yip *et al.* [345]	Smart Grids	✗	✗	Elect. Power	Linear Reg.	Anomaly	Simulation	Good
R. B. Diddigi *et al.* [85]	✗	WSN	✗	Net.Sensors	POMDP+MCarlos	Hybrid	Simulation	Good
N. Nissim *et al.* [238]	Network	✗	✗	Malw. Samples	active learn.	Anomaly	Simulation	Good
Wenjuan Li *et al.* [195]	✗	WSN	✗	RT.Traffic	MAS+ML	Hybrid	Testbed.	Good
C.J. Fung *et al.* [106]	Network	✗	✗	Audit Logs	MAS	Hybrid	Simulation	Good
M. S. R. *et al.* [267]	Smart Grids	✗	✗	Audit+PMU data	MAS	Hybrid	Simulation	Good

where $l(\bullet, \bullet)$ is a loss function. The optimal w is obtained, when the derivative of $E_{\mathcal{D}}(w)$ is null, i.e. $\nabla E_{\mathcal{D}}(w) = 0$. There are numerous classification methods including Support Vector Machines, Artificial Neural Networks, Naive Bayes, and Decision Trees.

a) Support Vector Machine. Support Vector Machines (SVMs) [70] are maximum margin linear classifiers formally defined by separating hyperplanes. For a given labeled training data set, the SVM algorithm outputs an optimal hyperplane able to classify a new data set. However, SVMs take a long time for training when the dataset is very large [24].

Theory. SVMs rely on the concept of linear separators that divides a set of examples into classes [36, 301]. $\mathcal{D}$ is linear separable if there exists a halfspace, (w, b), such that y_i =sign$(w^T \phi(x_i) + b)$, $\forall\, i$. In other words, the condition can be rewritten as follows,

$$\forall\, i \in \{1, ..., n\}, y_i(w^T \phi(x_i) + b) > 0$$

This condition involves multiple solutions that depend on the arbitrary initial values chosen for (w, b) and classify training datasets exactly. As there are multiple solutions, SVMs use the concept of margin. The margins of the separator is the distance between support vectors. Support vectors are the examples closest to the separators. The margin is given by the perpendicular distance (r_i) to the closest point x_i from the data set,

$$r_i = \frac{y_i(w^T \phi(x_i) + b)}{||w||}$$

The optimization problem of the parameters (w, b) to maximize the margin is defined by

$$\arg \max_{w,b} \min_{i} \; r_i$$

This problem can be reformulated into a much simpler one by assuming that $r_i.||w|| >= 1$. This one is to maximize $1/||w||$, i.e. minimize $||w||^2$ and it is given by

$$\arg \min_{w,b} \; \frac{1}{2}||w||^2$$

which is a quadratic problem. However, this approach assumes that training sets are linearly separable which is not always the case. To deal with non-linear separable training sets, relaxation variables (slack variables) denoted ξ_i ($\xi_i \geq 0$) allow the constraint $r_i.||w|| >= 1$ to be violated. The constraint is then reformulated to $r_i.||w|| >= 1 - \xi_i$ and the optimization problem is of the form,

$$\arg \min_{w,b,\xi} \left\{ \frac{1}{2}||w||^2 \right\} + C \sum_{i=1}^{n} \xi_i$$

where $C > 0$ handles the trade-off between the slack variable penalty and the margin.

Review. In cloud computing, M. Watson *et al.* [330] introduced an online cloud anomaly detection approach based on the one-class SVM. The authors evaluated the effectiveness of their approach using a controlled cloud testbed built on KVM hypervisors (under Linux). In the cloud testbed, they simulated DoS attacks and recent malware samples like Kelihos and multiple variants of Zeus. The proposed model was able to detect these attacks with a DR over 90%.

In Cloud computing, M. Al-Rubaie *et al.* [4] proposed a SVM-based attack reconstruction model to gain access to users' accounts on Active Authentication (AA) Systems. The system under attack consisted of an AA server for biometric authentication and a client app to gain access to the cloud services (System Model). An attacker used the client app to access victim's cloud data (Adversary model). During biometric authentication, the authors extracted feature vectors of the system such as Location, Shape, Time, and Pressure. Afterwards, the authors reconstructed raw gesture data from the user's authentication profiles (full-profile attack) and the decision value returned by SVM sample testing. The reconstruction was based on a numerical

algorithm that estimates the original raw data from summary statistics in the feature vector using SVMs and a randomized algorithm that adds Gaussian noise on data and generates randomized raw data. Through several experiments, the randomized approach performed well for attack reconstruction.

In MANETs, S. Doss *et al.* [86] combined authenticated routing-based framework and support vector machine (SVM) to detect Jelly Fish attacks. The authenticated routing-based framework relied on a hierarchical trust evaluation of the node property so that only trusted nodes are selected for route path construction. A node in MANET can evaluate trust of neighboring nodes using various metrics such as Packet Delay, Packet Availability and Packet Forwarding. Each has a priority level assigned (High, Low, Medium). With SVM, the nodes cannot deviate from their expected behavior. When any change occurs, it is immediately notified and that node leaves the routing path. The authors validated their approach using NS2-simulator and two datasets. The first one with category labels and the second one from MIT Lincoln Lab without category labels. Through several experiments, the proposed model was able to detect Jelly Fish attacks and achieved good performance in terms of throughput (22.67% on average), packet delivery ratio (13.34% on average), dropped packet ratio (43% on average) and end-to-end delay (51%).

b) Artificial Neural Network. An Artificial Neural Network (ANN) [212] is a computational model inspired from biological neural networks. It consists of an interconnected group of artificial neurons and processes information using a connectionist approach to computation [136]. Basically, the model has three layers: (a) a layer of input neurons that receives data from the real world, (b) a hidden layer consisting of various interconnected structures which process the inputs into something that the output layer can use and (c) an output layer that sends information directly to the outside world, to another computer process. ANNs are able to adapt and learn through back propagation by computing the cost of paths and adjusting the weight of neurons. However, they can take a long time to train and are difficult to scale [298].

Theory. There are several ANN types such as feedforward NNs, self-organizing NNs (e.g. Kohonen network) and temporal NNs (e.g. time-delay NN). We restrict our attention to feedforward NNs. A feedforward NN [301] is a directed acyclic graph $G = (V, E)$, where V are neurons and E are edges, with a weight function over the

edges, $w : E \to \mathbb{R}$. A neuron is represented by a scalar function $f : \mathbb{R} \to \mathbb{R}$, also called the activation function of the neuron. f can be an identity function i.e. $f(\bullet) = \mathbb{1}_{[\bullet>0]}$, a sign function i.e. $f(\bullet)$ =sign$(\bullet)$, sigmoid functions (e.g. $f(\bullet) = 1/(1+e^{-\bullet})$, $f(\bullet) = tanh(\bullet)$), or a non-saturation function i.e. $f(\bullet)$ =max$(0, \bullet)$. A basic feedforward NN has three layers: an input layer, a hidden layer and an output layer. In the input layer, given n input variables $(x_i)_{i\in\{1,..,n\}}$, n linear combinations of variables are made, in the form [36],

$$c_j = \sum_{i=1}^{n} {w_{ji}}^{(1)} \phi_i(x_i) + w_{j0}^{(1)}$$

where $j \in \{1, ..., p\}$, p is the number of linear combination of the input layer, c_j are activations, $\phi_i(x_i) = x_i$ as it is a linear combination, ${w_{ji}}^{(1)}$ are weights of the first layer, and ${w_{j0}}^{(1)}$ are biases of the first layer. In the hidden layer, c_j are transformed using non linear activation function f and it returns hidden units d_j, i.e. $d_j = f(c_j)$. d_j are linearly combined to obtain output unit activations c_k of the form,

$$c_k = \sum_{j=1}^{p} {w_{kj}}^{(2)} d_j + w_{k0}^{(2)}$$

where $k \in \{1, ..., m\}$, m is the number of outputs, ${w_{kj}}^{(2)}$ are weights of the second layer and $w_{k0}^{(2)}$ are biases of the second layer. Each c_k is transformed using a sigmoid activation function σ, such that $y_k(x, w) = \sigma(c_k)$. The overall NN function is given by

$$y_k(x, w) = \sigma\left(\sum_{j=1}^{p} {w_{kj}}^{(2)} f\left(\sum_{i=1}^{n} {w_{ji}}^{(1)} \phi_i(x) + w_{j0}^{(1)}\right) + w_{k0}^{(2)}\right)$$

For m separate binary classifications, each output k is a binary class label $y_k \in \{-1, 1\}$, $k \in \{1, ..., m\}$. The probability that a given input x_i is in the first class is given by $y_k(x_i, w)$. Otherwise, it is given by $1 - y_k(x_i, w)$. If the class labels are independent, given the input vector, the error $E_{\mathcal{D}}(w)$ is of the form,

$$E_{\mathcal{D}}(w) = -\sum_{i=1}^{n}\sum_{k=1}^{m} [y_{ik} \ln(y_k(x_i, w)) + (1 - y_{ik}) \ln(1 - y_k(x_i, w))]$$

and denotes the cross entropy, i.e. the negative logarithm of the conditional distribution of the targets. For a multiclass classification problem, each input x_i is assigned

to one of m exclusive classes. The probability that x_i is in the class k (the softmax function) is defined by $y_k(x_i, w) = e^{c_k(x_i,w)} / \sum_{l=1}^{m} e^{c_l(x_i,w)}$. The cross entropy $E_{\mathcal{D}}(w)$ is given by

$$E_{\mathcal{D}}(w) = \sum_{i=1}^{n} E_i(w) = -\sum_{i=1}^{n} \sum_{k=1}^{m} y_{ki} \ln y_k(x_i, w)$$

Review. In MANETs, A. S. Sadiq *et al.* [280] proposed an ANN modified with a PSO-based heuristic algorithm to detect Smurf and Neptune attacks. The PSO-based heuristic algorithm is inspired by magnetic field theory in physics that deals with attraction between particles scattered in the search space. Each magnetic particle has a measure of mass and magnetic field due to its fitness. The acceptable magnetic particles are those with the higher mass and higher magnetic field. The hybrid model consisted of three main entities: default gateway, network flow collector (server), and the ANN analyzer. In the training phase, the authors used the information gain as feature selection and trained the ANN classifier using the KDD cup99 dataset. Through experiments, the proposed model achieved a high DR of 99.5%.

In Medical Healthcare Systems, M. Barni *et al.* proposed a secure ANN-based ECG (Electrocardiogram) processing approach that encrypts patient data and classifies ECG diseases such as Normal Sinus Rythm, Atrial Premature Contraction, and Ventricular Tachycardia. Upon starting, ECG signals are preprocessed and autoregressive features (e.g. heart rate, peak-to-peak) are extracted from ingoing heart beats and encrypted. The ANN-classifier is then trained using the encrypted features. During training, the ANN-classifier securely decrypts features and the model is trained using the Levenberg-Marquardt learning algorithm [344]. Next, a Decision Tree is applied on outputs for medical diagnosis. Overall, the proposed model was able to classify diseases with a high accuracy larger than 86.30% and preserve the security of patient data.

A. Saied *et al.* [16] implemented an ANN-based IDS to detect known and unknown DDoS attacks. The authors generated datasets, using a realistic corporate safe environment where they launched different UDP, ICMP and TCP DDoS attacks while normal traffic was flowing through the network. The datasets were organized and structured in a qualified Java Neural Network Simulator (JNNS) format to train the ANN model. Afterwards, they evaluated their approach in two phases. Firstly,

the authors integrated the model in Snort (Snort-AI) and tested it on known and unknown DDoS traffic. They were able to achieve an ACC of 98% than existing NIDSs like Snort. Secondly, the authors tested their model using old datasets (attacks between 2000 and 2003) and new datasets (attacks between 2000 and 2013). They achieved an average ACC of 95% (unknown DDoS) and 100% (known DDoS).

c) Naive Bayes classifier. A Naive Bayes classifier [190] is a classification algorithm for two-class (binary) and multi-class classification problems. It is based on the Bayesian theorem with an assumption of independence among predictors [219], by assuming that the presence of a particular feature in a class is unrelated to the presence of any other feature.

Theory. The Naive Bayes classifier relies on the Bayes theorem. Let us assume m classes $c_1, c_2, ..., c_m$ and a new example x_i. The classifier assigns x_i to the class c_j, when c_j has the highest posterior probability conditioned on x_i,

$$P(c_j|x_i) > P(c_k|x_i)$$

for all $k \in \{1, ..., m\}$ and $k \neq j$. The classification problem consists in maximizing $P(c_j|x_i)$. Given the Bayes theorem, we have

$$\begin{aligned} c_j &= \arg\max_{c_j} P(c_j|x_i) \\ &= \arg\max_{c_j} \frac{P(x_i|c_j)P(c_j)}{P(x_i)} \end{aligned}$$

where $P(x_i)$ remains constant for all classes, $P(c_j) = n_j/n$ i.e. the number n_j of occurrences of c_j over the size n of the training set, and $P(x_i|c_j)$ is estimated from the training set. For example, if $P(x_i|c_j)$ is Gaussian,

$$P(x_i|c_j) = \frac{1}{\sqrt{2\pi}\sigma_{c_j}} e^{-\frac{(x-\mu_{c_j})^2}{2{\sigma_{c_j}}^2}}$$

where σ_{c_j} (variance) and μ_{c_j} (mean) are computed from all occurrences of c_j in the

training set. $P(x_i)$ being constant, only $P(x_i|c_j)P(c_j)$ should be maximized,

$$c_j = \arg\max_{c_j} P(x_i|c_j)P(c_j)$$

Let us assume n conditionally-independent examples $x_1, ..., x_n$ that we want to predict if they are in the class c_j. The classification problem is given by

$$\begin{aligned} c_j &= \arg\max_{c_j} P(x_1, ..., x_n|c_j)P(c_j) \\ &= \arg\max_{c_j} \prod_{i=1}^{n} P(x_i|c_j)P(c_j) \end{aligned}$$

as $P(x_1, ..., x_n) = \prod_{i=1}^{n} P(x_i)$ remains constant for all classes.

Review. M. Swarnkar *et al.* [289] proposed a content anomaly detection approach (OCPAD) based on an one-class Multinomial Naive Bayes classifier for identifying suspicious payload contents (HTTP attacks) in network packets. The OCPAD model is trained using the likelihood of each sequence's occurrence in a payload of known non-malicious packets. The authors stored the likelihood range of these sequence's occurrences (generated by OCPAD) in a probability tree structure, which is used to classify new payload contents as normal or malicious. In the testing phase, they used an academic dataset of one million HTTP packets and achieved good performance than other methods, with a high DR (up to 100%) and a low FPR (less than 0.6%).

In IoT, H. Haddad Pajouh *et al.* [137] proposed a classification model that consists of two components. The first one uses component analysis and linear discriminate analysis for dimension reduction. The second one combines Naive Bayes and K-Nearest Neighbor to detect DoS, Probe, U2R and R2L attacks. In the training phase, the model is trained and tested using the KDD cup99 dataset. As a result, the proposed model achieved a high DR of 84.84% and a low FPR of 5.21% on average.

d) Decision Tree. A Decision Tree (DT) [258] is a non-parametric supervised learning method used for classification and regression. It is a set of splitting decision rules used to segment the predictor space into a number of simple regions. DTs are known to be accurate because their cost is logarithmic in the number of samples used to train the model. However, DT learners can create over-complex trees that

do not generalize the data well (overfitting) [265]. It becomes necessary to set the maximum depth of the tree and the minimum number of samples required at a leaf node (pruning) [265].

Theory. A DT [301] is a prediction function, $\mathcal{D} \rightarrow \mathcal{C}$, that predicts the label $c \in \mathcal{C}$ associated with an example $x \in \mathcal{D}$. We restrict to the binary classification i.e. $\mathcal{C} = \{0, 1\}$ and $\mathcal{D} = \mathbb{R}^d$. At each node from the root to leaf path, the successor child is based on splitting rules or one of features of x. A splitting basis could be a threshold function,

$$\mathbb{1}_{[x_i \lesseqgtr \gamma]} = \begin{cases} 1 \ \text{if} \, x_i \lesseqgtr \gamma \\ 0 \ \text{else} \end{cases}$$

where $\gamma \in \mathbb{R}$ is the threshold and x_i ($i \in \{d\}$) is the most relevant feature of x. Another splitting basis could be decision rules R_k of the form IF-THEN with the semantic below (premiss/conclusion).

$$R_k \ \frac{x_1 \lesseqgtr \gamma_1, ..., x_d \lesseqgtr \gamma_d}{c}$$

where $x_1, ..., x_d$ are features of x, $\gamma_1, ..., \gamma_d$ are thresholds and c the label assigned to x. Note that rule conditions could rely on strings instead of reals i.e. $x \in \Sigma^d$, $\gamma \in \Sigma$ where Σ is an alphabet.

To build a decision tree, the algorithm ID3 (Iterative Dichotomizer 3) is often used. ID3 begins from the root and assigns it a label based on a majority vote among all labels over the dataset. Iteratively, from the root to childs, ID3 measures the information gain that quantifies the effect of splitting children. The information gain $G(\mathcal{D}, x)$ is based on the decrease in entropy after the dataset $\mathcal{D}$ is split on one attribute x_i of x. It is defined by

$$G(\mathcal{D}, x) = H[\mathcal{D}] - \sum_{x_i \in x} P(x_i) H[x_i]$$

where $H[\mathcal{D}] = \sum_{x \in \mathcal{D}} -P(x) \log_2(P(x))$ is the entropy on the overall dataset such that $P(x)$ the probability of occurrences of x in D, $P(x_i)$ the probability of occurrences of x_i in x, and $H[x_i]$ is the entropy on the feature x_i of x.

Amon all possible splits, ID3 chooses the one with the largest information gain as

the decision node, divides the dataset again using a splitting basis and repeats the same process recursively until all the examples are classified.

Review. In Smart Grids, U. Adhikari *et al.* [14] proposed an intrusion detection model based on Hoeffding adaptive trees [33] augmented with the drift detection method [118] and adaptive windowing [32] for the binary-class or multi-class classification of power system contengencies and cyber-attacks. The authors trained the model using a dataset containing 45 classes of cyber-power contingencies. After tests, the proposed model achieved a high ACC greater than 98% for binary-class and 94% for multi-class classification.

In Zigbee, H. J. Patel *et al.* [255] proposed a DT-based approach that relies on non-parametric Random Forest (RndF) and Multi-Class AdaBoost (MCA) ensemble classifiers to enhance Radio Frequency-Distinct Native Attribute (RF-DNA) fingerprinting in ZigBee device authentication. In the training phase, the authors used a pre-classification Kolmogorov-Smirnoff Test (KS-Test), a post-classification RndF feature relevance ranking, and a Generalized Relevance Learning Vector Quantization-Improved (GRLVQI) feature relevance ranking [149] for dimensional reduction. Through several tests, the proposed model achieved a higher DR of 90% on a benchmark and correctly rejected 31 of 36 rogue access attempts in ZigBee devices.

In RFIDs, K. Bu *et al.* [53] uses breadth first tree traversal to detect cloning attacks and unreconciled collisions in large anonymous RFID systems. Three protocols where proposed such as BASE, DeClone and DeClone+ for probabilistic and deterministic clone detection. Through simulations, DeClone+ offered a faster clone detection when the clone ratio was high.

J. Kevric *et al.* [172] proposed a hybrid classifier model that used the random tree and NBTree algorithms to classify the simulated network traffic as normal or malicious. They trained and tested the hybrid classifier model using the NSL-KDD dataset. Through the experiments, the authors were able to achieve a high ACC of 89.24% than other individual classifiers.

1.4.1.2 Regression-based techniques

Regression [111] is a supervised statistical approach used for continuous output values given continuous and/or categorical predictors. Regression models can be

divided in three main categories [36]: linear regression, nonlinear regression and generalized linear models. Linear regression [181] models the relationship between two variables x (explanatory variable) and y (dependent variable) by fitting a linear equation to observed data. While, non-linear regression models the relationship between the two variables by fitting a nonlinear equation to observed data (see Fig. 1.6). Generalized linear models [241] are extensions of previous regression models where the dependent variable y can be any member of a set of distributions called the exponential family (e.g. Normal, Poisson, Binomial). Most known examples are Logistic Regression (Logit) and Poisson Regression.

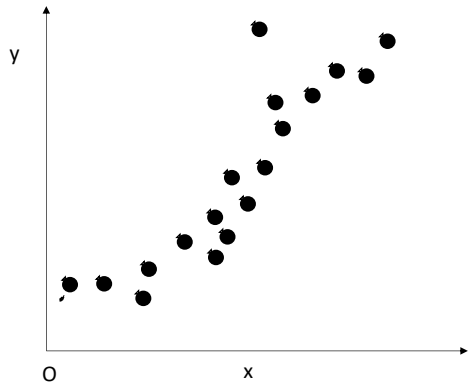


Figure 1.6 – Regression: An Overview

Theory. Let us consider the prediction function in Sect. 1.4.1.1. For linear regression, given an example x, $\phi(x) = x$ and the prediction function takes the form,

$$y(w, x) = w^T x + b$$

where $x = (x_1, ..., x_n)^T$, $w = (w_1, ..., w_n)^T$ are weights, and $b = w_0$ the bias. The optimization problem is to find optimal w that minimize the error $E_{\mathcal{D}}(w)$. $E_{\mathcal{D}}(w)$ can be deduced using the maximum likelihood or the maximum posterior. The maximum likelihood assumes that y are known and w are unknown. It maximizes the conditional probability $P(y|x, w) = P(y_1, .., y_n|x_1, .., x_n, w)$. If $(x_i, y_i)_{1 \leq i \leq n}$ are independent, $P(y_1, .., y_n|x_1, .., x_n, w) = \prod_{i=1}^{n} P(y_i|x_i, w)$. By assuming examples are Gaussian (i.e. $P(y_i|x_i, w) = \mathcal{N}(y_i|y(x_i, w), \sigma^2)$), the optimization problem is given by

the expression,

$$\begin{aligned} w &= \arg\max_{w} P(y|x,w) \\ &= \arg\max_{w} \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{1}{2}\left(\frac{y_i - y(x_i,w)}{\sigma}\right)^2} \end{aligned}$$

when we apply a logarithm function, the optimization problem takes the form,

$$w = \arg\min_{w} \underbrace{\sum_{i=1}^{n} \frac{(y_i - y(x_i,w))^2}{2}}_{E_{\mathcal{D}}(w)}$$

The maximum posterior consists in maximizing the conditional probability $P(w|x,y)$ i.e. maximizing $P(y|x,w).P(w)$ (Bayes theorem) as $P(x,y)$ is constant for w. Following the same process below, the optimization problem is given by

$$w = \arg\min_{w} \underbrace{\sum_{i=1}^{n} \frac{(y_i - y(x_i,w))^2}{2} + \lambda \frac{w.w^T}{2}}_{E_{\mathcal{D}}(w)}$$

where λ is the regularization parameter depending of σ^2. It is arbitrarily fixed or randomly-chosen during the cross-validation. For non-linear regression, $\phi(x)$ is non linear. The prediction function is given by the expression,

$$y(w,x) = w^T\phi(x) + b = W^T\tilde{\phi}(x)$$

where $W = (b, w_1, .., w_n)^T$ and $\tilde{\phi}(x) = (1, \phi_1(x), .., \phi_n(x))^T$. For instance, $\tilde{\phi}(x) = (1, x^1, x^2, .., x^n)^T$ in the case of the polynomial regression. Generically, the optimization problem (maximum posterior) can be rewritten as follows,

$$W = \arg\min_{W} \underbrace{\sum_{i=0}^{n} \frac{(y_i - W^T\tilde{\phi}(x_i))^2}{2} + \lambda \frac{W.W^T}{2}}_{E_{\mathcal{D}}(W)}$$

Review. In Smart Grids, S. Chin Yip *et al.* [345] designed two linear regression-based algorithms to study consumers' energy utilization behaviour, in order to detect energy theft (frauds) and defective smart meters. The authors introduced, in the model, indicators of energy theft i.e. detection coefficients (e.g. if indicator = 0 then "normal utilization" else "energy theft"), and categorical variables to identify the periods and locations of energy frauds and faulty smart meters. Simulations showed that the proposed approach successfully detected all the fraudulent consumers.

In power systems, J. Zhang *et al.* [347] proposed a multiple linear regression (MLR) classification model to identify false data injection (FDI) attacks. The attack model is based on the attacker's knowledge and capability by the assumption that the attacker has perfect knowledge of the topology/historical load data and also has enough historical data to perform MLR. The MLR model learns relationships from the external network and the attacker sub-network using the power transfer distribution factor (PTDF) matrix. The PTDF matrix captures all effects in the external network through the pseudo-boundary injections. The MLR model also used DC optimal power flow (OPF) features such as power balance, thermal limit, and generation limit constraints only in the outside area. In the testing phase, the model was evaluated using the IEEE 24-bus RTS and IEEE 118-bus systems. As a result, the attacker can overload transmission lines with the proposed model.

1.4.1.3 Clustering techniques

Clustering [157] is an unsupervised learning approach that partitions unlabeled data into groups in order to understand their structure (see Fig. 1.7). There are many clustering techniques such as k-Means clustering, graph-based clustering, hierarchical clustering, density-based clustering. Most have many applications in biomedical, finance and industry. However, they still have some problems in terms of parameter setting, executing efficiency and clustering capabilities such as low efficiency, algorithm complexity and noise-removing difficulty [353].

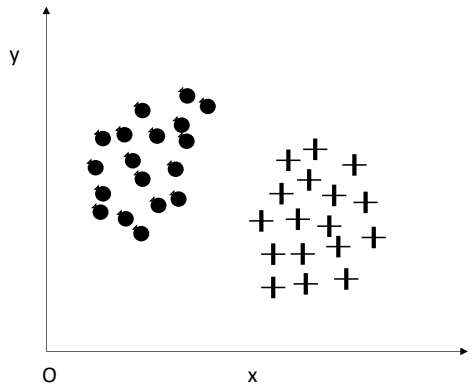


Figure 1.7 – Clustering: An Overview

Clustering considers a long sequence of examples $\mathcal{D} = \{x_1, x_2, ..., x_N\}$ without their targets. The aim consists in minimizing the distance between similar examples and maximizing the distance between dissimilar examples. The distance is a symmetric function $d : \mathcal{D} \times \mathcal{D} \to \mathbb{R}^+$, such that $d(x, x) = 0, \forall\, x \in \mathcal{D}$. Given an input parameter k, Clustering partitions $\mathcal{D}$ into k-clusters. The set of clusters $C = \{C_1, ..., C_k\}$ is defined such that $\bigcup_{i=1}^{k} C_i = \mathcal{D}$ and $\forall\, i \neq j, C_i \cap C_j = \varnothing$. In soft clustering, a probability $p_i(x)$ is assigned to each example $x \in \mathcal{D}$, where $p_i(x)$ is the probability that x belongs to cluster C_i.

a) K-Means clustering. K-mean clustering is an unsupervised learning method that partitions data into k clusters, represented by their centers. The center of each cluster k is caelculated as the mean of all the instances belonging to that cluster.

Theory. Given an input parameter k and p-examples $x_1, x_2, ..., x_n \in \mathcal{D}$ $(n \leq N)$, K-means clustering consists in finding k cluster centers $\mu_1, \mu_2, ..., \mu_k \in \mathcal{D}$ that minimize the quadratic function [36],

$$\sum_{i=1}^{n} \sum_{j=1}^{k} I_{ij} ||x_i - \mu_j||^2$$

where I_{ij} has value 1 when x_i is assigned to cluster j (i.e. $j = \arg\min_j ||x_i - \mu_j||^2$) and 0 otherwise. The cluster center μ_j is given by

$$\mu_j = \frac{\sum_{i=1}^{n} I_{ij} x_i}{\sum_{i=1}^{n} I_{ij}}$$

where $\sum_{i=1}^{n} I_{ij}$ is the number of points assigned to cluster j.

Review. In Big Data, K. Peng *et al.* [249] proposed a clustering method for IDS based on Mini Batch K-means combined with principal component analysis to detect complex attacks. The proposed approach has two main steps: Preprocessing and Detection. The first one digitizes and normalizes strings in the dataset by using replace function. Next, a principal component analysis (PCA) method is applied on processed data to reduce their dimensionality and the results is forwarded to the Mini Batch Kmeans clustering model for intrusion detection. The authors evaluated their approach using the KDDCUP99 dataset and compared Kmeans, Kmeans with PCA, single Mini Batch Kmeans. As a result, the model is effective and efficient (very low clustering time).

b) Graph-based clustering (GBC) is a method that uses graphs to produce clusters. A GBC is divided in two categories [264]: between-graph (full) clustering that partitions a set of graphs into different clusters, and within-graph (single) that divides the nodes of a graph into clusters.

Theory. A sequence of data $x_1, ..., x_n \in \mathcal{D}$ is represented in a graph structure $G = (V, E)$, V are vertices and E are edges with weights. There are many graph structures such as k-nearest neighbor (kNN) and spanning trees. A spanning tree $ST = (V', E')$ is a tree that contains all the vertices of the graph G. A minimum spanning tree MST [346] is a spanning tree with the smallest total weight. A k-MST is a graph consisting of k MSTs such that k-MST$= \bigcup_{i=1}^{k} \mathsf{MST}_i$. The k-MST-based clustering partitions G into k MSTs to achieve clusters. It relies on the weight of an edge of G and iteratively removes the edges of the biggest weight until a partition of G is achieved. In spite of removing edges, other mechanisms such as split-and-merge could be used [349]. During splitting, initial cluster centers are generated from the k-MST graph and K-means clustering is applied to achieve a partition. Each partition is adjusted so that it belongs to a sub-tree of the MST. During merging, the clusters in partition are combined using a merge criteria to produce large clusters. The merge criteria depends on the degree of information in the MST [349]. It could be to select vertices $v_i \in V$ with maximum/minimum degree.

Review. In Big Data, C. Sudipta *et al.* [62] proposed a graph-based clustering method that used self-organizing map clustering and topology features of graph nodes (i.e. clustering coefficient, degree, weight, node betweenness) to efficiently detect

botnets. In the testing phase, the authors evaluated their approach using NetFlow features in the CTU-13 dataset. Results shown that the proposed model was able to isolate bots in clusters of small sizes while containing the majority of normal nodes in the same big cluster.

c) Hierarchical-based clustering [160] is a clustering method which outputs a hierarchy (structure) that is more informative than the unstructured set of clusters returned by other clustering methods. The method constructs the clusters by recursively partitioning the examples in either a top-down or bottom-up fashion [264]. In recent papers, many papers [253, 269] have integrated hierarchical clustering in different ways.

Theory. Hierarchical clustering relies on two main parameters [43]; the distance between clusters $D : \mathcal{D} \times \mathcal{D} \rightarrow \mathbb{R}^+$ and the stopping criterion θ as clusters going large when examples are recursively partitioned. Let $C_i \subseteq \mathcal{D}$ and $C_j \subseteq \mathcal{D}$ be two clusters. The distance $D(C_i, C_j)$ can be computed using a single-linkage, group-linkage, and complete-average. In single-linkage, D is the minimum distance between examples of the two clusters and it is of the form,

$$D(C_i, C_j) = \min_{x \in C_i, y \in C_j} d(x, y)$$

The group-linkage assumes that D is the average distance between each cluster example,

$$D(C_i, C_j) = \frac{1}{|C_i||C_j|} \sum_{x \in C_i, y \in C_j} d(x, y)$$

In complete-linkage, D is the maximum distance between examples of the two clusters,

$$D(C_i, C_j) = \min_{x \in C_i, y \in C_j} d(x, y)$$

Review. In Cognitive Radio WSNs, K. Rina *et al.* [269] developed a model based on K-medoids clustering and agglomerative hierarchical clustering to identify spectrum sensing data falsification attack. In this attack, a node falsifies its local sensing report before sending it to the fusion center. The goal being to damage the final sensing decision of the fusion center. Through numerical simulations, the proposed model was able to detect and isolate malicious nodes using the collection reports at the fusion

center.

In Big Data, M. S. Parwez *et al.* [253] implemented a big data analytic model that combined k-means and hierarchical clustering methods to analyze users' mobile network data. The authors tested the model and compared the obtained results i.e. detected anomalies with truth information in hand to identify regions of interest in the network i.e. specific behaviours like faults, user-anomalies and malicious activities. After that, they used these abnormal behaviours and normal data to show the impact of anomalies in data while training intelligent models.

d) Density-based clustering is a clustering approach where the dataset that belongs to each dense region (cluster) is drawn from a specific probability distribution [264]. An example of density-based clustering is DBSCAN (density-based spatial clustering of applications with noise) [90]. A DBSCAN discovers clusters of arbitrary shapes, in large spatial databases, by searching the neighborhood of each object in the database and checks whether it contains more than the minimum number of objects [264].

Theory. A density-based clustering [129, 180] assumes that the neighbourhood of a given radius ϵ ($\epsilon > 0$) should contain a minimum number of m examples. Given an example $x \in \mathcal{D}$, the density of the neighbourhood denoted $N_\epsilon(x)$ should exceed some threshold and it is given by

$$N_\epsilon(x) = \{y \in \mathcal{D}, d(x, y) \leq \epsilon\}$$

In a cluster, an example x follows three properties: directly density-reachability, density-reachability, and density-connectivity. x is directly density-reachable from an example y w.r.t. ϵ and m if $x \in N_\epsilon(y)$ and $N_\epsilon(y) \geq m$. Moreover, x is density-reachable from an example y w.r.t. ϵ and m if there is a sequence of examples $x_1, ..., x_n$ with $x_1 = y$, $x_n = x$ such that x_{i+1} is directly density-reachable from x_i. In addition, x is density-connected from an example y w.r.t. ϵ and m if there is an example z such that x and y are density-reachable from z w.r.t. ϵ and m.

Review. C. Zhang *et al.* [352] developed an IDS approach that used density peak clustering with support vector data description (SVDD), able to process high-dimensional data with non-uniform density and identify network intrusion. The proposed approach has five steps. The first step consists in generating training and test

sample set using KDD cup99 and UCI datasets. The second one performs the cutoff distance optimization based on the adjusted silhouette coefficient and outputs the best cutoff distance. The third one used the best cutoff distance to achieve clustering on training samples (k-clusters) using density peak clustering. The fourth step uses an improved Particle Swarm Optimization to optimize parameter for the next step. This one uses optimized parameters to train the support vector data description and produce k sub-hyper spheres. After several tests, the proposed model achieved a high F-score of 90.9% on average with the UCI dataset, a high Detection Rate of 95.2% and F-score greater than 87% on average with the KDD cup99 dataset.

1.4.1.4 Evolutionary computing and swarm intelligence techniques

Evolutionary computing (EC) [340] is an advanced area which studies and designs biologically-inspired algorithms (e.g. gene reproduction). Swarm intelligence takes inspiration from the social behaviors of insects and other animals to solve problems. EC methods perform optimizations and learning tasks with abilities to evolve for complex problems. The basis cycle of EC methods consists of 4 main steps [340] (see Fig. 1.8):

Populating. EC methods maintain a whole collection of candidate solutions (populations) simultaneously, to optimize and learn the problem. Basically, they create an initial population of individuals. An individual is a solution in a population. In genetics, an individual has its gene representation called its code.

Evaluation. EC methods compute the objective values of the individuals for preparing the fitness process.

Fitness and selection. EC methods use objective values to determine the fitness. The fittest individuals are then selected for the reproduction.

Reproduction. Individuals will undergo a number of variation operations (e.g. genetic mutation/crossover) to mimic biological changes (e.g. genetic gene changes), which is necessary for the solution space. As a result, new individuals are then created and reused for the evaluation process.

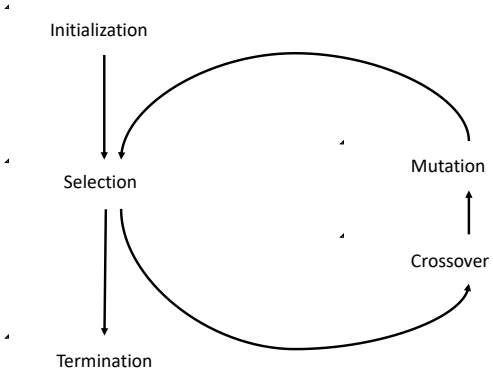


Figure 1.8 – Evolutionary computing cycle

In recent years, EC and swarm intelligence were successfully applied in various domains like engineering, medicine and transportation. Nonetheless, they may have some limitations [154]: (a) cannot guarantee finding the optimum solution in a finite amount of time, (b) need to tune various search parameters without proper guidance on how to set them for unknown problems, and (c) utilize too much exploration (computationally expensive) due to the use of the population-based search approach (greedy heuristic). Most known EC and swarm intelligence algorithms include Ant Colony Optimization (ACO), Artificial Bee Colony (ABO), Gravitational Search Algorithm (GSA), Genetic algorithm (GA), Memetic algorithm (MA), Particle Swarm Optimization (PSO), Self-organizing maps (SOM), Artificial Immune System (AIS), Cuckoo Search (CS) and Firefly algorithms.

a) Ant Colony Optimization (ACO). An ACO [74] is a meta-heuristic concept inspired from the foraging behavior of some ant species. Ants deposit pheromone on the ground in order to mark some favorable path that should be followed by other members of the colony. ACO exploits a similar mechanism to solve optimization problems. In recent years, an ACO has been successfully applied to improve the intrusion detection process in a target environment.

Theory. ACOs was introduced [74] to solve combinatorial optimization problems. A combinatorial optimization problem COP is defined by the tuple $\langle \mathcal{S}, f \rangle$, where $\mathcal{S}$ is the search space i.e. a finite set of solutions and $f : \mathcal{S} \rightarrow \mathbb{R}^+$ is the objective function that assigns a positive cost value to each of the solutions, the goal being to find a solution with the minimal cost. An ACO is a stochastic COP that relies on the

concept of pheromone model. A pheromone model is a vector $\mathcal{T}$ of pheromone trail parameters $\mathcal{T}_i$ that have pheromone values τ_i. An ACO model is the 3-tuple $\langle \mathscr{S}, \Omega, f \rangle$ where the solution space $\mathscr{S}$ is defined over a set of n pheromone trail parameters $\mathcal{T}_i^j \in T = \{\mathcal{T}_i^1, ..., \mathcal{T}_i^{|T|}\}$ with values $\tau_i^j \in D_i = \{\tau_i^1, ..., \tau_i^{|D_i|}\}, i \in \{1, ..., n\}$. An instantiation of a pheromone trail parameter is the assignment of a value τ_i^j to a pheromone trail parameter $\mathcal{T}_i^j$ i.e. $\mathcal{T}_i^j = \tau_i^j$. A globally optimal solution $s^* \in \mathscr{S}^* \subseteq \mathscr{S}$ satisfies the property $f(s^*) \leq f(s)$, for all $s \in \mathscr{S}$. Each solution s should satisfy the constraints $\omega \in \Omega$.

A generic ACO algorithm is described below. ACO algorithm starts by initializing all pheromone values to a constant value $c > 0$. Next, it probabilistically constructs solutions using heuristic information and transition probabilities. If valid solutions are found, a local search is applied to improve the solutions constructed by ants. Then, a pheromone value update rule is applied to increase the pheromone values from high quality solutions as follows,

$$\tau_i^j \leftarrow (1 - \lambda).\tau_i^j + \lambda \,. \sum_{s \in \mathscr{S}_{upd}} F(s)$$

where $\mathscr{S}_{upd} \subseteq \mathscr{S}$ is the input solutions to update, λ is the evaporation rate that uniformly decreases all the pheromone values, and $F : \mathscr{S} \rightarrow \mathbb{R}$ is a function such that $f(s_1) < f(s_2) \Rightarrow F(s_1) \geq F(s_2)$ for all $s_1 \neq s_2 \in \mathscr{S}$. All the process below is repeated until the optimal solution s^* is found.

Review. In VANETs, M. H. Eiza et al. [143] developed a secure and reliable multi-constrained Quality of Service (QoS) aware routing approach based on Ant Colony Optimisation. The proposed approach computed feasible routes subject to multiple QoS constraints and considered the topological properties of VANETs including variable communication link quality and frequent link breakages. While searching for feasible routes, ants selected their next hop when they arrived at intermediate nodes using state transition rules. Note that ants only traversed more reliable links to avoid traversing vulnerable links that are highly prone to breakage. A level of pheromone was associated to communication links between vehicles. When a vehicle received routing control messages, it used the received information to assign a pheromone value to each link w.r.t. QoS constraints. In the testing phase, the authors used

the OMNET++ 4.3 network simulator. The proposed model achieved a high packet delivery ratio and efficiently identified feasible routes.

Recently, F.H. Botes *et al.* [42] used an ACO-based decision tree classifier called Ant Tree Miner (AMT) to classify the dynamic behaviour of attacks as normal or abnormal. In the training phase, the authors train the AMT model with 20% of the NSL-KDD training dataset. In the testing phase, the AMT model has been evaluated using the NSL-KDD Test-21 dataset (records with difficulty of 21) [308] and outperformed the J48 Decision Tree classifier by 3% in detecting DoS attacks. Moreover, the FPR was reduced to 0% using the AMT model.

E.K. Varma *et al.* [318] combine ACO and Fuzzy entropy algorithms to search the best smallest network traffic features, which can be used to detect different type of real-time intrusion attacks like DDoS, account hijacking and probe. To evaluate the selected features, the authors run different classifiers (J48, Random Tree or Forest) using Weka tool and two standard benchmark datasets: Iris and Cleveland. After feature reduction, the model generation time was 37.19% faster than the full feature dataset and the Random Forest's accuracy increased by 0.27%.

b) Artificial Bee Colony. An Artificial Bee Colony (ABC) [166] is a swarm-based meta-heuristic technique, inspired by the intelligent foraging behavior of honey bees. In ABC, artificial bees fly around in a multidimensional search space to discover the places of food sources (good solutions for a given problem) with high nectar amount by changing food positions (individuals) to find the one (optimal solution) with the highest nectar.

Theory. Let $\mathcal{S}$ be the number of food sources (solutions) around the hive. The optimization problem [75] consists in finding the maximum nectar amount of the food source at the position $\theta \in \mathbb{R}^d$. The nectar amount of the i-th food source at θ_i is represented by the objective function $F(\theta_i)$. Once the food source is chosen, bees share their information with other bees (onlookers) within the hive and the ones probabilistically select one of the food sources. The probability P_i that the food source located at θ_i will be chosen by a bee is of the form,

$$P_i = \frac{F(\theta_i)}{\sum_{j=1}^{S} F(\theta_j)}$$

Onlookers go to the location of the food source θ_i using their belief P_i and determine a neighbour food source to take its nectar by moving from position $\theta_i(c)$ to $\theta_i(c+1)$ where c is the cycle. The position of the selected neighbour food source takes the form,

$$\theta_i(c+1) = \theta_i(c) \pm \phi_i(c)$$

where $i \in \{1, .., |S|\}$, $\phi_i(c)$ is a randomly produced step to find a food source with more nectar around θ_i. If $F(\theta_i(c+1)) > F(\theta_i(c))$, the bee goes to the hive, share information to others and $\phi_i(c) \leftarrow \phi_i(c+1)$ otherwise $\phi_i(c)$ is not changed. When the food source i cannot be improved, the food source at ϕ_i is abandoned and the bee (scout) search for a new food source.

Review. In WSNs, M. Korczynski *et al.* [170] proposed a bee-inspired method using DIAMOND to detect ealier distributed attacks such SYN flooding attacks and TCP portscan activity. DIAMOND is a distributed coordination framework that builds coordination overlay networks on top of physical networks and dynamically combines direct observations of traditional localized/centralized network IDS (NIDS) with knowledge exchanged with other coordinating nodes to automatically detect anomalies of the physical networks. The authors developed their own prototype communication protocol using OpenFlow and evaluated it using the Mininet 2.0 network emulator. During the testing phase, the network traffic was captured from the trans-Pacific line and was labelled by the MAWI working group as anomalous or normal using an advanced graph-based method that combines responses from independent anomaly detectors built from principal component analysis (PCA), the gamma distribution, the Kullback-Leibler divergence, and the Hough transform. Overall, the proposed model achieves a high ACC of 94.51%.

In Smart Grids, L. Yang *et al.* [341] combined extreme learning machine (ELM) and ABC methods to identify data injection attacks. Firstly, an autoencoder was used to reduce the dimensionality of the measurement data. Then, ABC finds optimal parameters i.e. number of hidden nodes and input weights. The optimal parameters are then sent to the ELM classifier for detection. In the testing phase, the authors used IEEE 118-bus, IEEE 14-bus systems and the MATPOWER tool to simulate the operation of the power network. Overall, the proposed model gave good results with a medium FPR of 11.06% and a high DR of 86.8% on average.

c) Gravitational Search Algorithm. A Gravitational Search Algorithm (GSA) is a heuristic optimization algorithm based on the law of gravity and mass interactions [270]. In GSA, the searcher agents are a collection of masses that interact with each other using the Newtonian gravity and the laws of motion. These agents have four parameters [6]: position, inertial mass, active gravitational mass, and passive gravitational mass.

Theory. Let us consider n agents acting on the search space $\mathcal{S}$. The position of an agent i is given by $x = (x_i^1, .., x_i^k, .., x_i^{|S|}), i \in \{1, .., n\}$, where x_i^k is the position of the agent i in the k-th dimension, and $|S|$ is the dimension of the search space $\mathcal{S}$. Once the search space is identified, GSA generates an initial population and evaluates the fitness function for each agent in the population. First, GSA updates the gravitational constant G. G has initial value G_0 and it is exponentially reduced in the time to control the search accuracy,

$$G(t) = G_0 e^{-\lambda \frac{t}{T}}$$

where G_0, λ are constant given during initialization and T the number of iteration. Let $f_i : \mathbb{R}^+ \to \mathbb{R}$ be the fitness function of the agent i and $f_i(t)$ the fitness value of the agent i at time t. For a minimization problem, the worst fitness value $w(t)$ and the best $b(t)$ are expressed as follows,

$$\begin{cases} b(t) = \min_{j \in \{1,..,n\}} f_j(t) \\ w(t) = \max_{j \in \{1,..,n\}} f_j(t) \end{cases}$$

Given a fitness value of the agent i, GSA updates its gravitational mass m_i and inertial mass M_i as follows,

$$\begin{cases} m_i(t) = \dfrac{f_i(t) - w(t)}{b(t) - w(t)} \\ M_i(t) = \dfrac{m_i(t)}{\sum_{j=1}^{n} m_j(t)} \end{cases}$$

6. https://www.igi-global.com/dictionary/gravitational-search-algorithm/48381

m_i and M_i are then used to compute the acceleration γ_i^k of the agent i in time t in the k-th dimension. Then, GSA updates the agent velocity v_i^k and position x_i^k in time t in the k-th dimension,

$$\begin{cases} v_i^k(t+1) = r_i.v_i^k(t) + \gamma_i^k(t) \\ x_i^k(t+1) = x_i^k(t) + v_i^k(t+1) \end{cases}$$

where r_i is a random number in the interval $[0, 1]$, $\gamma_i^k(t)$ the acceleration of the agent i defined by

$$\gamma_i^k(t) = \frac{\sum_{j=1, j\neq i}^{n} r_j.F_{ij}^k(t)}{M_i(t)}$$

where $F_{ij}^k = (G_0 M_{pi} M_{aj})/(d_{ij}^k)^2$ is an external force exerted by an agent j on an agent i in time t in the k-th dimension, such that d_{ij}^k is the distance between the two agents, M_{pi} and M_{aj} are respectively the passive gravitational mass of the agent i and the active gravitational mass of the agent j. GSA repeats the process below until stopping criteria met.

Review. In WSNs, D. Tirtharaj *et al.* [73] combined GSAs and ANNs to detect malicious network traffic. In the training phase, the ANN model is first trained using a single GSA and next with a combination of GSA and Particle Swarm Optimization [340](GSPSO-ANN). The resulting GS-ANN and GSPSO-ANN models are then tested using the NSL-KDD dataset and compared to existing models such as decision trees and ANNs based on genetic algorithms. The proposed model achieved a DR of 94.9% for GS-ANN and 98.13 % for GSPSO-ANN.

B. Hamid *et al.* [50] proposed a hybrid feature selection approach using a binary gravitational search algorithm (BSGA) and mutual information (MI). The proposed method has two layers of optimization: (a) an outer optimization layer that uses the BGSA technique to find an optimal feature subset and (b) an inner optimization layer that prunes the feature dataset to improve the previous layer. The authors evaluated their model using SVM as a fast binary classifier and tested it with the NSL-KDD datatset. After experiments, the MI-BGSA model outperformed some existing feature selection approaches with a high ACC (88.362%) and a relatively low FPR (8.887%).

d) Genetic algorithm. Genetic algorithms (GAs) are optimization algorithms that

repeatedly modify a population of individual solutions [340]. Individuals have two properties: (a) its location (chromosome composed with genes), and (b) its quality (fitness value). GAs randomly select individuals from the current population and use them as parents to produce via crossover and recombination operations, the children for the next generation [340]. As the generation cycle is repeated, the population evolves toward an optimal solution.

Theory. GAs are inspired from behavior of chromosomes. A chromosome is a string of genes. Each gene stores one from alleles (a finite number of values). Formally, a chromosome [285] is a string $C = c_1c_2...c_p \in \Sigma^p$, where Σ is the alphabet and p is the length of the chromosome C. A chromosome C is similar to another one w.r.t. a schema, which is the same length than C. A schema is a string $S = s_1s_2...s_p \in (\Sigma \cup \{\epsilon\})^p$, ϵ is an empty symbol. C matches a schema S when $c_i = \epsilon \wedge c_i = s_i$ for all $i \in \{1, ..., p\}$. A GA starts by initializing a population $\mathcal{S}$ with n chromosomes $C_1, ..., C_n$. Then, GA selects an individual C_i from the population $\mathcal{S}$ with above-average fitness using a probability $P_s(C_i)$ defined by

$$P_s(C_i) = \frac{f(C_i)}{\sum_{j=1}^{n} f(C_j)}$$

where $f : \Sigma^p \rightarrow \mathbb{R}$ is the fitness function. Individuals with above-average fitness tend to receive low-average fitness. Next, GA applies a crossover operation on two chromosomes $C_i = c_1^i...c_p^i$ and $C_j = c_1^j...c_q^j$, where $p = |C_i|$ and $q = |C_j|$. A simple crossover operation generates two offspring chromosomes $C_i' = c_1^i...c_k^ic_{k+1}^j...c_r^j$ and $C_j' = c_1^j...c_k^jc_{k+1}^i...c_r^i$, where k is a random number in $\{1, ..., r-1\}$ and r is the length of each generated chromosome. After crossover operation, chromosomes are mutated. The mutation consists in randomly choosing a gene and swapping its value to increase the structural variability of $\mathcal{S}$. The process below is repeated until a criterion is fulfilled.

Review. D. Papamartzivanos *et al.* [251] used GAs and Decision Trees to generate new accurate detection rules able to classify common and zero-day attacks. The proposed system, called Dendron, was evaluated and tested using three datasets: KDDCup 99 (ACC=98.5%, FPR=0.75%), NSL-KDD (ACC=97.55%,FPR=1.08%) and UNSW-NB15 (ACC=84.33%,FPR=2.61%). The best result was obtained using

KDDcup99.

M.R. Gauthama *et al.* [275] proposed a hypergraph-based GA method (HG-GA) for parameter setting and feature selection in the SVM method. The authors evaluated their model using NSL-KDD and compared it to existing methods such as GA-SVM, PSO-SVM, BGSA-SVM, Random Forest, and Bayesian Networks. They achieved good performance with a high DR (95.32%) and a low FPR (0.83%).

e) Memetic algorithm. A Memetic algorithm (MA) is a corporative heuristic approach based on a *meme* [210] notion in cultural evolution. In MAs, a population of optimizing agents cooperate and complete by communicating with each other and using different heuristics: approximation algorithms, local search methods or specialized recombination operators [210].

Theory. A MA relies on the concept of agents that represents a tentative solution for the problem [7]. Let $\mathcal{A}$ be a search space, an agent i is denoted $A_i \in \mathcal{A}$. Agents undergo multiple competition and mutual cooperation operations mimicking the behaviors of living beings from a same species. First, MA generates a population by selecting agents based on their goodness. The goodness of an individual is evaluated using the information from the fitness function $f : \mathcal{A} \to \mathbb{R}$. The selection decision is made upon a probability $P(A_i) = f(A_i)/\sum_{A_j \in \mathcal{A}} f(A_j)$. Resulting agents are sent for reproduction. The reproduction consists in creating new agents from the older ones. To create new ones, MA uses recombination operations such as crossovers and cooperation. Afterwards, the agents mutate using local-improvers. Local improvers start from an initial generated agent $A_0 \in \mathcal{A}$ and iteratively uses at each step a transition based on the neighborhood of the current agent A_{curr}. The transitions going to preferable agents are accepted (i.e. $\mathsf{curr} \leftarrow i$) when $f(A_i) < f(A_{\mathsf{curr}})$. The operation terminates by the means of a criterion.

Review. In WSNs, C. Chen *et al.* [65] used a MA model to maximize sensing coverage while achieving energy efficiency at the same time. This is particularly necessary for intrusion detection nodes that require a full coverage at any time. The proposed approach consisted of a MA-based scheduling strategy and a heuristic recursive algorithm (HRA). In the MA-based scheduling, the MA is used with a dynamic genetic structure to achieve a consecutive exploring process and create a maximal

7. http://www.lcc.uma.es/ ccottap/papers/IntroMAs.pdf

number of disjoint sets of sensor nodes. In addition, a sleep schedule is built for other off-duty nodes to conserve energy by using the power saving mode of IEEE 802.15.4, such that only the minimum nodes are active to cooperatively provide full coverage without redundancy. On the other hand, the HRA is used to deal with the dynamic-coverage-maintenance problem so that the loss of sensing coverage can be recovered. The authors evaluated their approach using a WSN testbed and simulations on the MATLAB tool. As a result, the proposed model was able to maximize sensing coverage while achieving energy efficiency at the same time.

f) Particle Swarm Optimization. A Particle Swarm Optimization (PSO) [167] is a population-based stochastic optimization method that mimics the ability of a bird flock to fly synchronously, change directions suddenly, scatter, and regroup [340]. A PSO uses two main notions: (a) velocity to describe the movement of birds and (b) particle (in the sense of physics) to their method. Like GAs, PSOs are initialized with populations of random solutions and search the optimal solutions by updating generations. Frequently, PSOs are used in intrusion detection systems for the selection of the best features to accurately detect network or host attacks.

Theory. A PSO is initialized with a set of random particles $\mathcal{X}$ and searches for optima by updating generations. For each iteration, the position $x_i(t)$ of each particle i is updated in time t based on its velocity $v_i(t)$, the best solution x_i^{best} achieved by the i-th particle (fitness) and the best global position x^{gbest} in the search space $\mathcal{X}$. Over time, the particle converges/clusters together around one or several optima through a combination of exploration and exploitation of good position in $\mathcal{X}$ [167]. The velocity v_i and the position x_i of the i-th particle is updated as follows,

$$\begin{cases} v_i(t+1) = v_i(t) + \lambda_1\, r_i(x_i^{\mathsf{best}} - x_i(t)) + \lambda_2\, r_i(x^{\mathsf{gbest}} - x_i(t))) \\ x_i(t+1) = x_i(t) + v_i(t) \end{cases}$$

where $v_i(t+1)$ is the new velocity of the particle i, λ_1 and λ_2 are the weighting coefficients for the best and global best positions respectively, r_i is a random number belonging in the interval $[0, 1]$, x_i^{best} is the i-th best position of the the particle p_i, $x_i(t)$ is the i-th position of the particle i, x^{gbest} is the global best position in $\mathcal{X}$, and $x_i(t+1)$ is the new position of the particle i. The process below is repeated when a minimum error criteria is achieved.

Review. Recently, M. H. Ali *et al.* [2] developed a model based on fast learning network and PSO-based optimization for zero-day attack detection. A fast learning network is a double parallel forward neural network (DPFNN). A DFNN is defined such that the re-coded information from the hidden nodes, along with the information from the input nodes is fed into the output nodes. A PSO-based optimization selects the best weight's values and the number of neurons needed in the hidden layer to achieve better accuracy. The authors trained and tested their approach using the KDD cup99 dataset. Overall, the proposed model achieved a high ACC of 99% on average.

Recently, Z. Yi *et al.* [342] proposed a feature selection approach that used Particle Swarm Optimization to detect Java Script malwares. Firstly, the authors used ASTParser (an abstract syntax tree parser) to extract ASTs in code fragments of downloaded malwares. After that, AST nodes are saved as features consisting of two parts: the node's structure (e.g. FunctionInvocation) and its content (e.g. XMLHttpDownload). In the training phase, the authors collected 742 malicious and 854 benign scripts using a malicious javascript dataset [193] and VirusTotal public API, and extracted AST nodes of scripts before applying PSO-based feature selection on. In the testing phase, the model gave good results with a high ACC (over 98%).

g) Self-organizing map. A self-organizing map (SOM) is an unsupervised learning ANN algorithm used to reduce dimensionality of feature vectors [178]. It has the special property of effectively creating spatially organized *internal representations* of various features of input signals and their abstractions [178].

Theory. A SOM relies on a set of neurons, represented in 2-D grid to form a discrete topological mapping (sensory-to-cortex mapping) of an input space $\mathcal{X} \subseteq \mathbb{R}^d$. A weight vector w_i (such that $|w_i| = d$) is associated to each neuron i. A SOM algorithm [67] starts by randomly initializing n weights $w_1, ..., w_n$ of neurons $1, .., n$. At each time t, given an input $x(t) \in \mathcal{X}$, SOM selects the best matching unit $\omega(t)$ as follows,

$$\omega(t) = \arg\min_{k \in \mathcal{K}} ||x(t) - w_k(t)||^2$$

where $\mathcal{K} = \{1, .., n\}$ is a set of neuron indexes. Afterwards, SOM updates the weight

of the solution and its neighbours,

$$w_k(t+1) = w_k(t) + \lambda(t)\phi(\omega, k, t)(x(t) - w_k(t))$$

where $\phi(\omega, k, t) = e^{-\frac{||l_\omega - l_k||^2}{2\sigma(t)^2}}$ is a neighbourhood function and l_ω, l_k denotes the location vectors of neurons ω and k, respectively. $\sigma(t)$ is the effective range of the neighbourhood and $\lambda(t)$ is the learning rate, randomly chosen within range $[0, 1]$. The process below is repeated until SOM converges.

Review. In visual sensor networks, K. Huang *et al.* [152] use a hierarchical version of self-organizing map (HSOM) and pattern learning to detect jamming, node replication, MITM and DoS attacks. Firstly, the authors developed a traffic model to describe the dynamic properties of network traffic in VSNs. Afterwards, they used the model to extract the most relevant features of traffic patterns and fed them into a HSOM neural network for pattern learning and intrusion detection. During simulations, the authors generated a network traffic using Omnet++ and WVSNMODEL simulators. The proposed approach gave a high DR of 95.3%, low FPR of 4.1% and detection time less than 100ms.

h) Artificial Immune System. Artificial Immune Systems (AISs) are computationally intelligent techniques inspired from the biological immune systems (BISs). BISs are adaptive and complex systems that defend the body from invading pathogens [72]. They are formed of cells and molecules or non-self cells that represent the collective coordinated immune responses. Moreover, BISs have three main layers [6]: (a) an anatomic barrier which is the first line of defense in the body, (b) an innate immunity i.e. an unchanging mechanism that detects and destroys invading pathogens, and (c) an adaptive immunity that responds to previously unknown invading cells and builds a response to them.

Theory. AISs have multiple mechanisms including artificial negative/positive selection and artificial clonal selection [113]. An artificial negative selection (ANS) starts by creating a set of self strings $\mathcal{S}$. Then, it creates a set of randomly generated strings $\mathcal{R}_0$. For each string $R_0 \in \mathcal{R}_0$, ANS forms a detector set $\mathcal{R}$ of those that do not strongly match any $S \in \mathcal{S}$. It uses a matching function F that indicates the level

of similarity between the strings, such that $F(R_0, S) \geq \lambda$, where λ is a threshold. For each $R \in \mathcal{R}$, ANS verifies that there is no $S \in \mathcal{S}$ matches above the threshold. Next, this operation is repeated again while change detection of S is required.

In the artificial clonal selection (ACS), an initial population of antibodies $\mathcal{A}$ is randomly created. Then, ACS selects a subset $\mathcal{S}$ of the fittest antibodies from $\mathcal{A}_t$ using a fitness function f. For each antibody $s_i \in \mathcal{S}$, ACS creates a set of clones $\mathcal{C}_i \subseteq \mathcal{C}$, where $\mathcal{C}$ is the set of all clones. Next, ACS mutates each clone $c \in \mathcal{C}$ and adds resulting clones $\mathcal{C}'$ to $\mathcal{A}_t$. An ACS selects again a subset $\mathcal{S}'$ of the fittest antibodies from the resulting $\mathcal{A}_t$ and randomly generates a new population. All the best members of the new population are stored in $\mathcal{A}_{t+1}$. The whole process is repeated until a criterion is met.

Review. In WSNs, J. M. Vidal *et al.* [322] proposed an AIS model based on building networks of distributed sensors, only required for the monitored network to detect DoS attacks. Each sensor network (immune cell) detects intrusions and reacts against the observed threats (immune response). This is done by emulating the different immune responses, the establishment of quarantine areas and the construction of immune memory. During the testing phase, the authors tested their approach using fooding attacks generated from the DDoSIM tool, public datasets (KDD'99, CAIDA'07, CAIDA'08) and traffic samples gathered by the University Complutense of Madrid. Overall, their approach achieve good performance with a high ACC on all the datasets. In IoT, R. Roman *et al.* [273] proposed a proactive security architecture that used AIS to defend against malicious edge nodes. The AIS model allowed only authorized immune cells (e.g., virtual machines) to traverse edge nodes. It also analyzed the security and consistency of the IoT infrastructure.

i) Cuckoo search. A Cuckoo search (CS) [338] is a nature-inspired optimization strategy based on the cuckoo brood parasitic behaviour in combination with Levy flight behaviour of other birds and fruit flies. It consists for the female to lay of the nest of another species so that it ensures the brooding of the egg and feeding the young individual. The CS model is divided in three steps [338]: (a) each cuckoo lays one egg (a solution) at a time, and dumps its egg into a randomly chosen nest (population), (b) the best nests with high quality of eggs (optimal solutions) will

carry over to the next generations and (c) the number of available host nests is fixed (population size), and the egg laid by a cuckoo is discovered by the host bird with a given probability.

Theory. A CS generates an initial population of n host nests x_i, for all $i \in \{1, ..., n\}$. Afterwards, CS gets a cuckoo i randomly by Levy flights. When generating new solutions $x_i^{(t+1)}$ (a cuckoo i), the Levy-flights is performed as follows [338],

$$x_i^{(t+1)} = x_i^{(t)} + \alpha \oplus \mathsf{Levy}(\lambda)$$

where $\alpha > 0$ is the step size. The Levy flight provides a random walk through a Markov chain whose next/current state depends of the current location (i.e. $x^{(t)}$) and the transition probability (i.e. $\alpha \oplus \mathsf{Levy}(\lambda)$). The random step length follows a Levy distribution defined by

$$\mathsf{Levy} \sim u = t^{-\lambda}$$

where $\lambda \in]1, 3]$. In addition, CS evaluates the fitness F_i of cuckoo i by randomly choosing a nest amount of a cuckoo j and j is updated with i (new solution) when $F_i > F_j$.

Review. In SDNs, I. H. Abdulqadder *et al.* [19] developed a secure architecture based on chaotic secure hashing for user authentication, enhanced genetic algorithm (EGA) and Cuckoo Search to mitigate flow table overloading attacks, control plane saturation attacks and Byzantine attacks. The model included multiple controllers in the control plane, multiple switches in the data plane and a monitoring cloud server. EGA and CS assigned controllers (individuals) based on the available links, latency, controller load balancing and alternative multiple path. In the testing phase, the authors implemented the proposed model in the OMNeT++ simulator and evaluated its performance in terms of packet loss, end-to-end delay, throughput, latency, and bandwidth. Five actions were taken during detection: Alert (analyze the new incoming flow), Quarantine (isolate the new incoming flow), Block (block the flow), Discard (delete the new incoming flow) and Move (install the flow after analysis). Overall, the proposed model achieved a throughput of 250Mbps for five switches, a low end-to-end delay (max. 40ms), low packet loss (max. 30), and low bandwidth consumption (null) for a simulation duration of 50sec.

Recently, G. Kanaka Raju *et al.* [124] proposed a customized CS-based model that used the Hamming distance to find optimal attributes network transactions i.e. continuous process of user's activities. The selected optimal attributes of normal and attack records are used to build nests for Cuckoo search. During simulations, the authors trained and tested their model using the ISCX IDS dataset (70% for training, 30% for tests). They achieved good performance with a high ACC (97%).

j) Firefly algorithm. A Firefly algorithm (FA) [336] is a nature-inspired meta-heuristic optimization algorithm inspired by the flashing behaviour of fireflies. Firefly's flash acts as a signal system i.e. an oscillator that charges or discharges the light at regular intervals to attract other fireflies. A FA is defined as follows [336]: (a) all fireflies (population) are unisexual, so one firefly will be attracted to all other fireflies; (b) attractiveness is proportional to their brightness (objective function), and for any two fireflies, the less bright one will be attracted by the brighter one; however, the brightness can decrease as their distance increases; and (c) if there are no fireflies brighter than a given firefly, it will move randomly.

Theory. Let $\mathcal{S}$ be the search space, I the light intensity of a firefly and β its attractiveness. Each firefly $s \in \mathcal{S}$ (potential solution) is represented by its light intensity I [336]. $I(s)$ is proportional to the value of fitness function $f(s)$ and it exponentially decreases as follows,

$$I(s) = I_0 e^{-\lambda s^2}$$

where I_0 is the light intensity of the source (i.e. when $s = 0$) , λ is a given light absorption coefficient λ. The attractiveness $\beta(s)$ of a firefly is proportional to its light intensity value $I(s)$ and it is given by

$$\beta(s) = \beta_0 e^{-\lambda s^2}$$

where β_0 is the attractiveness at $s = 0$. A firefly s_i moves in the search space $\mathcal{S}$ to find a more attractive firefly s_j using the fitness function f. The movement of a firefly s_i is based on its current position i, the attraction to another more attractive firefly s_j at position j, and a random walk depending on a randomization parameter α and a random generated value $\epsilon_i \in [0, 1]$. ϵ_i follows a Gaussian distribution. The movement

of a firefly s_i is of the form,

$$s_i = s_i + \beta(r_{ij})(s_j - s_i) + \alpha\epsilon_i$$

where $r_{ij} = ||s_i - s_j||$ is the distance between the two fireflies s_i and s_j. Globally, FA initializes a population $\mathcal{S}_0 \subseteq \mathcal{S}$ and determines a new random value α. Next, FA evaluates $s_i \in \mathcal{S}$ using its fitness value $f(s)$ and sorts s w.r.t. $f(s)$. The best solution is saved and the process below is repeated again.

Review. Recently, S.A. Raza Shah *et al.* [292] combined single MOTS IDSs (snort, suricata) with anomaly detection methods such as SVM and firefly algorithm, SVM and fuzzy logic. In the testing phase, the authors used a test bed where the input network traffic was generated using open source network traffic generators like Ostinato, NMAP, and NPING. SVM and firefly algorithm achieved better performance (ACC= 95%, FPR=8.6%) than SVM and fuzzy logic.

K. Munivara Prasad *et al.* [254] proposed a real-time bio-inspired anomaly IDS (BARTD) that combined Firefly and Bat [335] algorithms to detect efficiently different kinds of Distributed DoS attacks: network/transport level DDoS, application level DDoS and flooding (reflexion, HTTP-based). During tests, the authors used JMETER to generate dataset and prepared data i.e. partitioned data into flood and normal for training. As a result, the BARTD model outperformed single existing approaches (Cuckoo Search, Firefly), with a high ACC over 95 %.

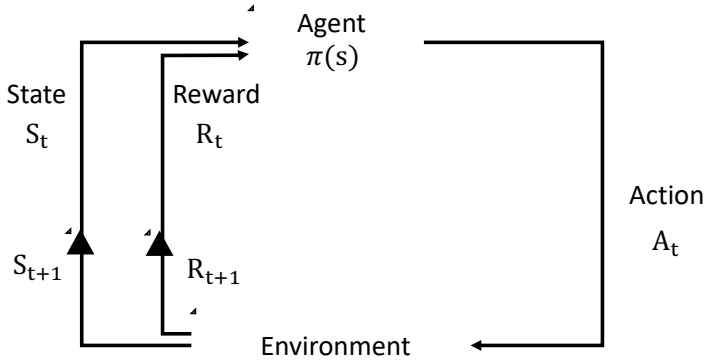


Figure 1.9 – Reinforcement Learning: Markov Decision Process

1.4.1.5 Reinforcement learning techniques

Reinforcement learning (RL) is a learning paradigm which learns to control a system so as to maximize a numerical performance measure that expresses a long-term objective [282]. RL achieves their goal in two main steps [282]: (a) the use of samples to compactly represent the dynamics of the control problem, and (b) the use of powerful function approximation methods to compactly represent value functions. Most known Reinforcement Learning algorithms are Markov Decision Process (MDP), Q-Learning and Bayesian Game Theory. RL has various application areas like Robotics, Finance, Industrial manufacturing and Power systems. However, RLs may have some limitations. They must learn the model of environment i.e. need to know where actions lead in order to evaluate actions and make decisions. In addition, they can be computationally difficult to optimally solve when the problem size increases (curse of dimensionality).

a) Bayesian Game Theory. A game consists of a number of players, a set of possible strategies for each player and the payoff function for each player [227]. A Bayesian game is a game which takes into account the nature as an additional player, and where the players have incomplete information on the other players but each player has a subjective probability distribution (beliefs) over the alternative possibilities [227]. A Bayesian game has different application areas including intrusion detection systems.

Theory. a Bayesian Game [227] is a tuple $\langle \mathcal{N}, \mathcal{A}, \Theta, P, u \rangle$ where $\mathcal{N} = \{1, .., n\}$ is a set of players, $\mathcal{A} = \{A_1, .., A_n\}$ is a set of actions where A_i is a pure strategy for player i, $\Theta = \{\Theta_1, .., \Theta_n\}$ is a set of signals or types where $\theta_i \in \Theta_i$ is a realization of types for player i, $P : \Theta \rightarrow [0, 1]$ is a joint probability distribution according to which types of players are drawn, and $u = \{u_1, ..., u_n\}$, where $u_i : \mathcal{A} \times \Theta \rightarrow \mathbb{R}$ is payoff function of player i. A pure strategy for player i is a map $s_i : \Theta_i \rightarrow A_i$ that prescribes an action for each type of player i. Each player i knows its type and evaluates its payoff according to a conditional distribution $P(\theta_{-i}|\theta_i)$, where $\theta_{-i} = \bigcup_{j=1, j\neq i}^{n} \theta_j$. A payoff of player i is given by

$$u_i(\tilde{s}_i, s_{-i}, \theta_i) = \sum_{\theta_{-i}} p(\theta_{-i}|\theta_i) u_i(\tilde{s}_i, s_{-i}(\theta_{-i}), \theta_i, \theta_{-i})$$

where $\tilde{s}_i \in A_i$. The best responses of player i to $s_{-i}(\theta_{-i})$ are given by

$$\mathsf{BNE}_i = \sum_{\theta_{-i}} p(\theta_{-i}|\theta_i) \left(\arg\max_{\tilde{s}_i \in A_i} u_i(\tilde{s}_i, s_{-i}(\theta_{-i}), \theta_i, \theta_{-i}) \right)$$

A strategy profile $s_i(\theta_i)$ of player i is a Bayesian Nash Equilibrium (BNE) if $s_i(\theta_i) \in \mathsf{BNE}_i$.

Review. In UAVNETs, H. Sedjelmaci *et al.* [300] proposed a Bayesian game model to defend the UAVNET against internal (e.g. false injection) and external (e.g. DoS) threats with a high detection accuracy and a low amount of intrusion detection data being exchanged between IDSs at each node. The Bayesian game was formulated between two couples: IDS vs attackers, Intrusion Ejection System (manages the ejection process of intrusive nodes) vs suspected node. The authors evaluated the model using urban scenario generated by the Simulation of Urban Mobility (SUMO) tool and achieved good performance with a high ACC between 96-98%.

In Smart Grids, K. Wang *et al.* [327] proposed a Bayesian honeypot game model to analyze the strategic interactions between the attackers (anti-honeypot strategy) and the defenders (honeypot strategy) using gathered information from honeypot sensors in an Advanced Metering Infrastructure (AMI). In the testing phase, the authors used a test bed that consisted of 4 servers, 10 honeypots, and 2 anti-honeypots. As a result, the model can reach a dynamic balance between detection rate and energy consumptions; it effectively detects DDoS attacks in AMI networks.

b) Markov Decision Process. A Markov Decision Process (MDP) is a memoryless stochastic process model [28] that consists of a set of states, a discrete set of actions, a conditional probability distribution which determines the transition from the current state to the next state, and a reward/cost function that outputs the immediate reward for given state and action (see Fig. 1.9). When the system state is not determined, MDPs are Partially Observable (POMDP). POMDPs [60] are defined by a set of partial observable actions, a set of observations, an observation distribution for each state and an initial state distribution. MDPs are widely used to solve complex problems in financial decision making (FDM), medical decision making (MDM), and industrial decision making (IDM). Recently, MDPs were also successfully applied in cybersecurity area precisely in intrusion detection and prevention systems.

Theory. A discounted Markov Decision Process [126] $\mathsf{MDP} = \langle \mathcal{S}, \mathcal{A}, P, R, \gamma \rangle$, where $\mathcal{S}$ is a finite or infinite set of all states, $\mathcal{A} = \{a_1, ..., a_n\}$ is the set of all actions, $P : \mathcal{S} \times \mathcal{A} \to \mathcal{P}(\mathcal{S})$ is the Markov transition kernel; such that $P(s'|s, a)$ is the probability distribution of the next state s' given any state s and action a, $R : \mathcal{S} \times \mathcal{A} \to \mathcal{P}(\mathbb{R})$ where $R(s'|s, a)$ is distribution of the immediate reward and γ is the discounted reward. A policy $\pi : \mathcal{S} \to \mathcal{P}(\mathcal{A})$ maps any state $s \in \mathcal{S}$ to a probability distribution $\pi(a|s)$ of taking action a when in state s. The value $V^\pi(s)$ of a state s under policy π, given a state-value function $V^\pi : \mathcal{S} \to \mathbb{R}$, is defined by

$$V^\pi(s) = \mathbb{E}\Bigg[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \Big| S_t = s, \\ A_t \sim \pi(A_t|S_t), S_{t+1} \sim \pi(S_{t+1}|S_t, A_t) \Bigg]$$

where t is any time step and $\mathbb{E}[\bullet]$ denotes the expected value of a random variable $\bullet$. Moreover, the value of taking action a in state s under a policy π, $Q^\pi(s, a)$, where the action-value function $Q^\pi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$. $Q^\pi(s, a)$ takes the form,

$$Q^\pi(s, a) = \mathbb{E}\Bigg[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \Big| S_t = s, A_t = a, \\ A_t \sim \pi(A_t|S_t), S_{t+1} \sim \pi(S_{t+1}|S_t, A_t) \Bigg]$$

Given any state s and action a, the optimal state-value function $V^*(s)$, the optimal action-value function $Q^*(s, a)$ and policy $\pi^*(s)$ are given by

$$\begin{cases} V^*(s) = \max_\pi V^\pi(s) \\ Q^*(s, a) = \max_\pi Q^\pi(s, a) \\ \pi^*(s) = \arg\max_a Q^*(s, a) \end{cases}$$

Review. In Smart Grids, H. Chen *et al.* [284] proposed a MDP-based approach

to model intruder's attack strategy and analyze the likelihood of attacks in power systems using an optimal intruder's attack strategy. The authors tested the model using IEEE 14-bus and IEEE 30-bus test case systems. Each test system used measuring devices to record the voltage phase of its located bus and the current phasor of incident lines. Finally, results have shown that the proposed approach successfully described the attack behaviour and vulnerabilities of the test systems.

In WSNs, R. B. Diddigi *et al.* [85] uses a POMDP-based model to track the intruder at each instant, while taking into account the energy consumption of sensors. The authors optimized the POMDP-based model using three RL algorithms: greedy algorithm, Monte Carlos tree search and hybrid (both of them). Next, they compared the performances of different algorithms, and concluded that the hybrid algorithm is more accurate for the mitigation of the state-action space explosion of the POMP-based model.

c) Q-Learning. Q-Learning is a model-free reinforcement learning that provides agents with the capability of learning to act optimally in Markovian domains by experiencing the consequence of actions, without requiring them to build maps of the domains [326]. Each agent performs the following steps [326]: (a) observes its current state, (b) selects and performs an action, (c) observes the next state, (d) receives an immediate reward, and (e) adjusts its Q-value using a learning factor.

Theory. The Q-Learning algorithm (QLA) is deduced from MDP. A QLA [326] relies on the state-value function V and the Q-value (i.e. the action-value function Q). The Q-value Q^* (resp. V^*) in the current state $(s, a)$is arranged in function of Q^* (resp. V^*) in the next state (s', a') as follows,

$$Q^*(s,a) = \sum_{s' \in \mathcal{S}} P(s'|s,a)[r + \gamma . \max_{a' \in \mathcal{A}_{s'}} Q^*(s',a')]$$

$$V^*(s) = \max_{a \in \mathcal{A}_s} \sum_{s' \in \mathcal{S}} P(s'|s,a)[r + \gamma . V^*(s')]$$

Initially, QLA arbitrarily sets an initial state-value function V_0. For each iteration i, QLA determines Q-values by computing Q_{i+1} (aka V_{i+1}) as follows,

$$Q_{i+1}(s,a) = \sum_{s' \in \mathcal{S}} P(s'|s,a)[r + \gamma . \max_{a \in \mathcal{A}_s} Q_i(s', a')]$$
$$V_{i+1}(s) = \max_{a \in \mathcal{A}_s} \sum_{s' \in \mathcal{S}} P(s'|s,a)[r + \gamma . V_i(s')]$$

This process is repeated until the criterion $|Q_{i+1} - Q_i| \geq \theta$ (aka $|V_{i+1} - V_i| \geq \theta$) is fulfilled.

Review. In Smart Grids, P. Jockar *et al.* [158] proposed a Q-Learning-based intrusion detection and prevention systems (HANIDPS) for ZigBee-based home area networks. The proposed system has two main modules: (a) a detection module that combines specification and anomaly methods using selected specifications from IEEE 802.15.4 standard, and (b) a prevention module that uses Q-Learning to find the best strategy against an attacker. The prevention module performs automatically preventive actions such as poofing prevention, interference avoidance and dropping malicious packets. In the testing phase, the authors simulated an IEEE 802.15.4 network that contains 6 TelosB motes (TPR2400), 4 air monitors, an attacker and a genuine node. As a result, they obtained an average DR of 92.5% and no FPR (0%) for the detection module. While in the prevention module, the average FPR is 13.805% and the average dynamic prevention performance is greater than 96%.

In UAVNETs, L. Xiao *et al.* [334] proposed a UAV power allocation strategy based on deep Q-Learning and WoL-PHC to address subjective smart attacks in a dynamic game such as jamming, spoofing and eavesdropping attacks. The attack game model provides a user-centric view of subjective smart attacks and consisted of two players: a smart attacker and the UAV system. A smart attacker makes subjective decisions to choose the attack type (jamming, spoofing, eavesdropping) without knowing the attack detection accuracy of the UAV system. The UAV system transmits power on multiple radio channels to resist smart attacks. Through simulations, the authors concluded that the proposed model increased the secrecy capacity and the utility of the UAV against subjective smart attackers.

1.4.1.6 Active learning techniques

Most known supervised and unsupervised learning techniques first gather a significant quantity of random data set from the world (underlying population distribution) and then induce a classifier or model [307, 315]. However, gathering data are often costly and time-consuming while we have limited resources for the collection [315]. A solution can be to gather only needed data from the world by asking queries, receiving responses, and asking further queries based upon the previous responses. This approach is called active learning. Both supervised and unsupervised techniques are used in active learning.

Theory. An active learning relies on the concept active learners. An active learner gathers information about the world by asking queries and receiving responses. It typically queries information in a pool (pool-based active learning) or in streaming (online learning) [315]. Let $\mathcal{I}$ be a measurable domain of elements and $\mathcal{R}$ a measurable domain of responses. We assume that for each $x_i \in \mathcal{I}$, there is probably a response $y_i \in \mathcal{R}$. An unlabeled pool is represented by the set $P = (x_1, ..., x_d) \in \mathcal{I}^d$. In pool-based active learning, an active learner [315] is a tuple $\langle f, q, D \rangle$, where $f : \mathcal{I} \to \{-1, 1\}$ is a classifier trained on the current set of labeled/unlabeled data $D \subseteq \mathcal{I}$. Given D, $q : \mathcal{I} \to \mathcal{R}$ is the query function that allows deciding which next example to query in P. After p-queries, active learners returns a set $\{(x_1, y_1), ..., (x_p, y_p)\}$. In online learning, given an infinite stream $\mathcal{S} \subseteq \mathcal{I}^\infty$, active learners query an unlabeled data $s \in \mathcal{S}$ and return its immediate label $f(s)$, indefinitely.

Review. Recently, N. Nissim *et al.* [238] proposed an active learning-based approach (ALDROID) to update anti-virus signature repository by automatically acquiring a maximum quantity of zero-day malwares and then, enhance the ALDROID's detection model using new malware samples. The authors tested their approach using the Android platform and Android's anti-virus softwares. As a result, the proposed system was able to acquire the largest number of zero-day malwares than existing heuristic approaches. Latterly, N. Nissim *et al.* integrated their approach in ALDOCX [231] to efficiently assist anti-virus vendors and improve the detection accuracy. ALDOCX automatically identifies and collects new docx files that are malicious and informative benign files. The authors evaluated their approach using 16,811 Mi-

crosoft Word files from Virus Total and Contagio, including 327 malicious and 16,484 benign files. Overall, ALDOCX achieved a high DR of 94.44% and a low FPR of 0.19%.

1.4.1.7 Deep learning techniques

Deep learning (DL) is inspired by the human brain's ability to learn from experience instinctively [127,135,136]. It allows computational models that are composed of multiple processing layers to learn representations of data with multiple levels of abstraction [189]. Unlike some learning methods, DL can be trained in an unsupervised, semi-supervised and supervised manner for various learning tasks. Recently, DLs have been successfully applied in various domains like speech recognition, computer vision, sound and image processing. It generated interest due to many reasons [76]: (a) reduction of the need for feature engineering, (b) absence of unsupervised pre-training and compression capabilities, and (c) easy adaptation to new problems relatively. However, DLs require large amounts of data and are often extremely computationally expensive to train (e.g. require expensive GPUs for complex models).

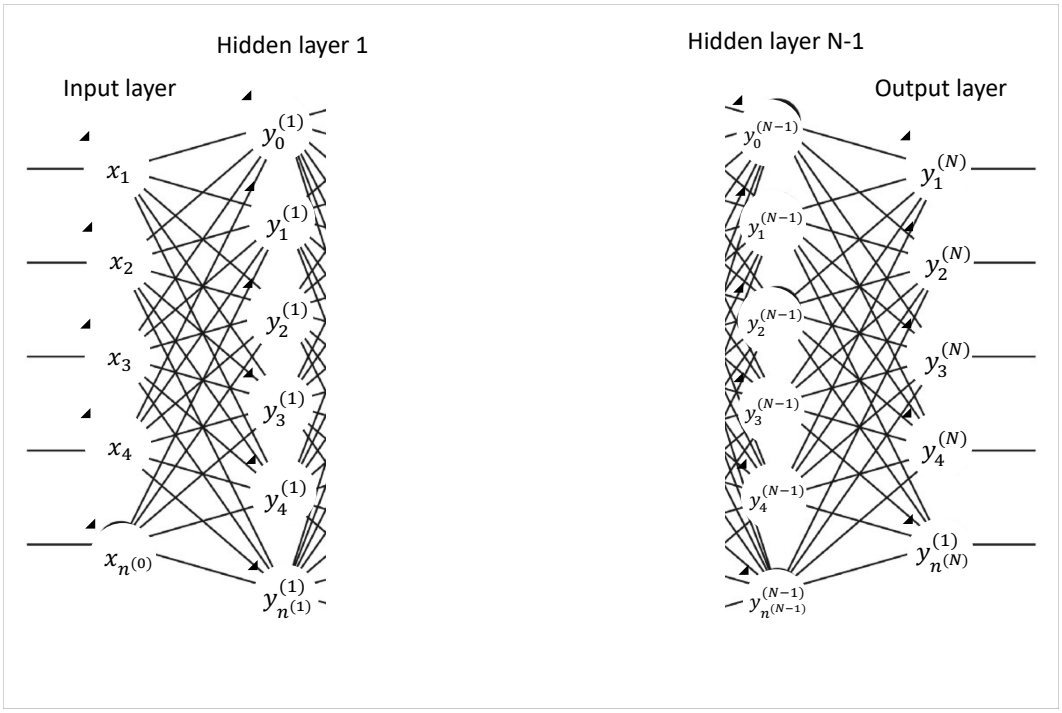


Figure 1.10 – Deep Neural Networks : An Overview

Theory. The first well-known DL technique is Deep Neural Network. A Deep Neural Network (DNN) is a composition of multiple hidden layers, typically more than three layers, with an input layer and an output layer. Let us consider a DL with N layers, consisting of $N-1$ hidden layers, each having $n^{(l)}$ hidden units ($l \in \{1, ..., N-1\}$), an input layer with $n^{(0)}$, and an output layer $n^{(N)}$ (see Fig.1.10). Given an activation function f (see Section 1.4.1.1-b)), the i-th unit within layer l is given by [20],

$$y_i^{(l)} = f(z_i^{(l)})$$
$$z_i^{(l)} = \sum_{k=1}^{n^{(l-1)}} w_{ik}^{(l)} y_j^{(l-1)} + w_{i0}^{(l)}$$

where $z^{(l)}$ and $y^{(l)}$ are vectors of the current inputs $z_i^{(l)}$ and the outputs $y_j^{(l-1)}$, respectively. $w_{ik}^{(l)}$ represents the weighted connection from the k-th unit in layer $l-1$ to the i-th unit in layer l ($w^{(l)}$ is the matrix), and $w_{i0}^{(l)}$ is the biais at l-th layer. $y(x, w) = y^{(N)}$ is the output vector of the DNN where w is the vector of all weights of the DNN. When f is a sigmoid function, a softmax function is applied on outputs and the cross entropy is similar to the one in Section 1.4.1.1-b) with $m = n^{(N)}$.

A naive update of the DNN is prone to vanishing and exploding gradients, making it impossible to train networks with multiple layers. Thus, it is necessary to use back-propagation in training DNNs, which relies on squashing activation functions such as sigmoid or hyperbolic tangent. Assuming a stochastic training, the input examples are randomly chosen and DNN's weights are updated using the training $E_i(w)$, given i input samples (see Section 1.4.1.1-b)). To avoid confusion, we will represent $E_i(w)$ by $E_q(w)$, given $n^{(0)}$ input samples $(x_0, ..., x_q, ..., x_{n^{(0)}})$. Upon starting, the DNN's weights are randomly initialized [114] within range

$$-\frac{\sqrt{6}}{\sqrt{n^{(l-1)} + n^{(l)}}} < w_{ij}^{(l)} < \frac{\sqrt{6}}{\sqrt{n^{(l-1)} + n^{(l)}}}$$

The initialization of DNN's weights is crucial. After the initialization, DNN's

weights are efficiently computed using the Gradient descent by iteratively updating weights as follows,

$$w[t+1] = w[t] - \gamma \frac{\partial E_q}{\partial w[t]}$$

where t is an iteration value, γ is the learning rate and $\partial E_q / \partial w[t]$ the weight error. As mentioned in [20], the back-propagation error algorithm (BPA) consists in propagating the input sample x_q through the DNN, by a chain derivation process until getting the current input and output of each unit. Errors $\delta_i^{(N)}$ for output units are computed as follows,

$$\delta_i^{(N)} = \frac{\partial E_q}{\partial y_i^{(N)}} f'(z_i^{(N)})$$

For all hidden layers l, BPA computes the error $\delta_i^{(l)}$, given by

$$\delta_i^{(l)} = f'(z_i^{(l)}) \sum_{k=1}^{n^{(l+1)}} w_{ik} \delta_k^{(l+1)}$$

and calculates the weight errors,

$$\frac{\partial E_q}{\partial w_{ji}^{(l)}} = \delta_j^{(l)} y_i^{(l-1)}$$

Usually, these training operations are costly, particularly, when training sets are very large. It becomes necessarily to parallelize data via asynchronous communication [274]. There is three levels of parallelisms [77]: model parallelism, data parallelism, and single-host-based multi-threading. In model parallelism, the N-layered DNN (i.e. $N-1$ hidden layers and one output) is partitioned into $N-1$ hosts. When the network is locally connected, there are communications across hosts. In data parallelism, the $n^{(0)}$ input samples are partitioned into M shards $(S_1, ..., S_M)$, where S_r is the r-th shard. The M shards forward data to M copies of the model (replicas) for training, respectively. The parameters of replicas $\mathcal{P} = (P_1, ..., P_M)$ are asynchronously sent to a set of hosts H. Each replica r computes the gradient on data from S_r, sends the parameter P_r to a host h and the one will update its own parameter copy P'_h and send it back to S_r. A single-host-based multi-threading uses cores of a single host to

perform matrix vector computations through multiple threads.

Other DL techniques include Deep Belief Networks (DBFs), Deep Boltzmann Machines (DBMs), Deep Autoencoders (DAs), Deep Recurrent Neural Networks (DRNNs), Deep Convolutional Neural Networks (DCNNs), Generative Adversarial Networks (GANs), and Deep Capsule Networks [286].

a) Deep Belief Networks. A Deep Belief Network (DBN) is a graphical model that learns to extract a deep hierarchical representation of the training data [147]. It is a stack of Restricted Boltzmann Machines (RBMs) [9] which are trained in a greedy manner. A RBM is an energy-based undirected generative method that models a distribution of observations over visible variables using binary hidden variables [194].

Review. In 5G mobile communications, L. F. Maimo *et al.* [104] developed a novel approach based on DBNs, long short-term recurrent neural networks (LSTM), and Stacked Autoencoders (SAE) to analyze network flows and detect new botnet attacks. The proposed approach has four modules: Virtualized Infrastructure (VI), Virtualized Network Functions (VNF), Management and Orchestration (MANO), and Operations and Business Support Systems (OSS/BSS). The first one virtualizes the physical resources and sends them to VNFs. The second one was divided in Anomaly Symptom Detection (ASD) and Network Anomaly Detection (NAD). The ASD component uses DBNs and SAE models with mini batchs to perform quick search of anomaly symptoms on network-flow aggregations. The NAD component collects timestamped symptoms associated to RAN (Radio Acess Network) and analyzes the timeline as well as the relationship among these symptoms using Long Short-Term Memory Recurrent Networks for anomaly detection. The MANO component ochestrates VNFs and the network slicing for supporting multi-tenancy. During the training phase, the authors selected 288 features from network flows (e.g. number of flows, entropy). Afterwards, the proposed model was tested using the CTU dataset and libraries such as Caffe2, PyTorch. Overall, the proposed achieved a high DR of 70.95% on average.

b) Deep Boltzmann Machines. A Deep Boltzmann Machine (DBM) is a graphical undirected generative model that learns complex internal representations from a large supply of unlabeled sensory inputs and limited labeled data [288]. Like Deep Belief Networks, DBMs are compositions or stacks of RBMs.

Review. In SDNs, S. Garg *et al.* [117] proposed a deep-learning-based anomaly

detection approach to identify suspicious flows in social multimedia. The proposed model consisted of an anomaly detection module that leverages RBMs and gradient descent-based SVM to detect the abnormal activities, and an end-to-end data delivery module to satisfy strict QoS requirements of the SDN, i.e. high bandwidth and low latency. The authors evaluated the proposed model using a Carnegie Mellon University (CMU)-based insider threat dataset. Through experiments, the proposed model gave good performance in terms of data delivery and successfully detected malicious events such as identity theft, profile cloning, and confidential data collection.

c) Deep Autoencoders. A Deep autoencoder (DAE) [150, 319] is a graphical feed-forward neural network which consists of two symmetrical Deep belief networks that typically have four or five shallow layers representing the *encoding* half of the network, and a second set of four or five layers that make up the *decoding* half. DAEs have different applications like information retrieval, dimensionality reduction and data compression.

Review. In Wi-Fi networks, M. E. Aminanto *et al.* [5] proposed a deep-feature extraction and selection based on stacked feature extraction and weighted feature selection to improve detection accuracy against impersonation, flooding, and injection attacks. The stacked feature extraction uses DAEs to transform the original features into a more meaningful representation by reconstructing its input. The weighted feature extraction uses ANNs, which is trained with two target classes only (normal and impersonation attack classes). The authors evaluated their approach using the Aegean Wi-Fi Intrusion Dataset (AWID). As a result, the proposed model achieved an ACC of 99.918% and a FPR of 0.012%.

d) Deep Recurrent Neural Networks. A Recurrent Neural Network (RNN) [146, 262] is a kind of neural network that includes weighted connections within a layer and a loop memory i.e. it can maintain information while processing new input; its decision at time (t-1) affects the new one at time t. Deep RNNs (DRNNs) are hierarchical representations of RNNs with some transition conditions [247]. In recent years, DRNs have been successfully applied on temporal-data-based systems that usually consider the history of the input such as machine translation, language modeling and generating text, speech recognition and intrusion detection systems.

Review. Y. Chuan *et al.* [64] developed an RNN-based intrusion detection system

to detect unknown attacks. In the training phase, the authors preprocessed data from the NSL-KDD dataset and extracted features to train the RNN-based model. The training of the model consists of two modules: (a) a forward propagation module that computes the output values, and (b) a back propagation module that updates weights with accumulated residual values. In the testing phase, the model is simulated using Theano-Weka and compared to existing ML techniques like J48 Decision Tree, ANN, Random Forest and SVM. As a result, the proposed model gave a better ACC (99.81% for KDDTRain+, 68.55% for KDDTest-21) than other methods.

e) Deep Convolutional Neural Networks. A Deep Convolutional Neural Network (DCNN) [105, 191] is a deep or generalized representation of ANNs that consists of a number of convolutional and subsampling layers, followed by one or more fully connected layers. DCNNs are usually used for image classification and, recently, they were successfully applied to intrusion detection systems.

Review. Z. Li *et al.* [198] applied the CNN model to intrusion detection systems, by converting the input dataset into an image before applying the CNN model on. The detection process consisted of five steps: (a) raw data capture, (b) extraction of the most relevant features, (c) data scaling using min-max normalization, (d) image representation, and (e) the feeding of the CNN model using received images for intrusion detection. In the testing phase, the authors used TensorFlow as deep learning framework, NSL-KDD to train the model and existing DCNN models for comparison such as ResNet 50 and GoogLeNet. As a result, the proposed model gave a high ACC for both ResNet 50 (81.57% on Test-21, 79.14% on Test+) and GoogLeNet (81.84% on Test-21, 77.04% on Test+).

f) Generative Adversarial Networks. A Generative Adversarial Network (GAN) [119] is a minimax two-player game model for estimating generative models via adversarial process, in which two models are trained simultaneously. The first model, called *generative*, is responsible for the capture of the model distribution and its training procedure consists in maximizing the probability of the second model making a mistake. Next, the second model, called *discriminative*, estimates the probability that a sample is from the model distribution rather than the first model. More recently, this framework was successfully tested in intrusion detection systems.

Review. In Cloud computing, N. Moustapha *et al.* [213] combined GAN and

Outlier Dirichlet Mixture (ODM-ADS) to process streaming data and detect advanced persistent threats in Fog networks. The proposal can self-adapt against injection attacks that targets the training phase and attempts to corrompt the learning process. It can also build a normal profile from network flows and identify deviations from the profile using Dirichlet Mixtures and GANs. The proposed model was evaluated using NSL-KDD and UNSW-NB15 datasets. As a result, the proposed model achieved a high ACC and high detection time than existing approaches.

As most ML models lack robustness to challenge real-world data of attackers, K. Grosse *et al.* [120] constructed a GAN-based robust attack against malware detection models. The authors' aim was to get as possible a high misclassification rate from detection models while optimizing their GAN-based detector model. The authors used DREBIN dataset to train malware detectors, and an example of an adversarial crafting algorithm to mislead trained malware detectors. As a result, they achieved misclassification rates over 69% against malware detectors. Inversely, their malware detector achieved a better ACC over 95% than existing detection models.

In Industrial and Control Systems, C. Feng *et al.* [101] proposed a real-time GAN-based learning method to conduct stealthy attacks using a Secure Water Treatment (SWaT) testbed [110]. This dataset was preprocessed using min-max scaling and used to train the stealthy attack GAN model. In the testing phase, the authors simulated attacks by randomly injecting malicious chemical measurement (HCl-hydrochloric acid, NaOCl-Sodium hypochlorite) in the begnin SWaT dataset and observed that compromised Programmable Logic Controller sensor channels, when injecting malicious measurements, have a high probability of conducting stealthy attacks.

1.4.2 Knowledge-based detection techniques

Basically, Knowledge-based systems (KBSs) [156] are computer systems that contain stored knowledge and solve problems like humans would. KBSs are often assimilated to System Experts (ESs) while ESs are just KBSs applied in a specific field. KBSs have three main components [88]: knowledge base (KB), inference engine (IE) and user interface (see Fig. 1.11). The knowledge base (KB) is a managed collection of structured or unstructured knowledge that includes storage and retrieval of

information via queries. Generally, the knowledge base is specified by a human expert of the target domain. The inference engine is a component that infers on the knowledge base using reasoning mechanisms (e.g. backward chaining, forward chaining, similitude-based) to make decision. Finally, the user interface dialogues with the backend of KBS for consultations, suggestions, and configurations.

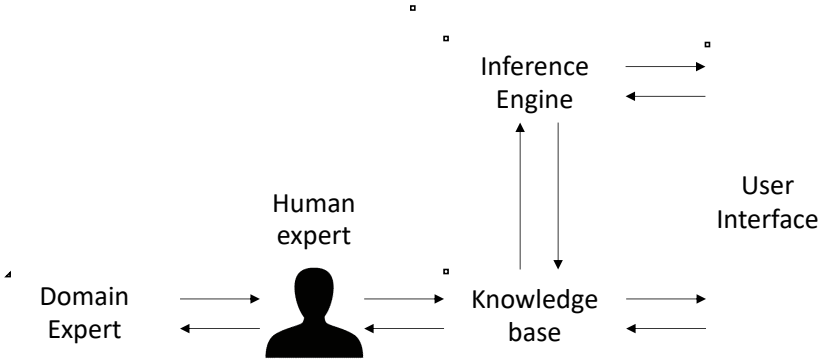


Figure 1.11 – Knowledge-based systems: An overview

KBSs have various advantages including [88]: (a) reduction of human intervention, (b) consistency of answers, (c) explanation of solutions, (d) efficient and cost effective. But they also present some limitations [88]: (a) restricted domain of expertise, (b) lack of learning ability as they depend of the human expert who expresses and articulates knowledges, and (c) cannot reason on the basis of human intuition or even of common sense. Recently, different Knowledge base reasoning approaches were proposed to challenging cyber threats in a given environment like Rule-based Reasoning (RBR), Case-based Reasoning (CBR), Ontology-based Reasoning (OBR), Model-based Reasoning (MBR), and Fuzzy-based Reasoning (FBR) detailed hereafter.

Theory. The knowledge base is an essential component in KBSs. There are various knowledge representation schemes including logic-based representation, procedural and structured representation. In logic-based representation, the knowledge relies on formal languages including First-order Logic [295] for an explicit representation of system data, B/VDM/Z [1,161] for modelling system data and states, and CSP/CCS [225,250] for modelling system behaviors and interactions.

In procedural representation, the knowledge is expressed as a set of instructions, often called rules in production systems. The rules are of the form

$$\text{if } P_1, ..., P_i \text{ then } G_1, ..., G_j$$

, where premises $P_1, ..., P_i$ are computed in order to achieve goals $G_1, ..., G_j$. The semantic of rules in the first-order predicate is given by

$$\begin{aligned} &\forall\, X_1, ..., X_{n_j}, Y_{1,1}, ..., Y_{1,m_1}, ..., Y_{i,1}, ..., Y_{i,m_i} \bullet \\ &P_1(Y_{1,1}, ..., Y_{1,m_1}) \wedge ... \wedge P_i(Y_{i,1}, ..., Y_{i,m_i}) \rightarrow \\ &\qquad G_1(X_1, ..., Y_{n_1}) \wedge ... \wedge G_j(X_{n_1}, ..., Y_{n_j}) \end{aligned}$$

where P_i and G_j are predicate names of the rule, X_j and $Y_{i,k}$ are variables. More often, it is simplified using the clause,

$$\frac{P_1, ..., P_i}{G_1, ..., G_j}$$

On the other hand, the structured representation expresses and structures complex knowledge i.e. objects or concepts of the target domain and relations between them. A well-known example of structured representation is Ontology. A simple ontology [222] $O = \langle C, P, A, I \rangle$ consists of a conceptualization C for each possible world that includes concepts and instances, a set of properties and relations P, a set of axioms and rules A, and an interpretation model I, such that

$$\begin{aligned} C &= (\bigcup_{t \in \text{Type}} C_t) \sqcup (\bigcup_{i \in \text{Type}} I_i) \\ P &= (\bigcup_{p \in \text{Type}} P_p) \sqcup (\bigcup_{e \in \text{Type}} Rel_e) \\ A &= (\bigcup_{a \in \text{Type}} A_a) \sqcup (\bigcup_{r \in \text{Type}} R_r) \end{aligned}$$

The properties and relationships are respectively given by P=(D:DataType, O:Object, T:Transitive, F:Functional)) and $Rel = (\equiv, \subset, \cap)$. For example, let consider three concepts: Shape, Circle, and Square. Circle $\subset$ Shape, Square $\subset$ Shape, and Circle $\cap$

Square = ∅ are relationships between concepts.

1.4.2.1 Rule-based Reasoning

Rule-based Reasoning (RBR) [48] is the most widely used reasoning paradigm which consists of two main components: (a) a set of "IF-THEN" rules representing domain knowledge (knowledge base), and (b) an inference engine containing some domain-independent inference mechanisms such as backward or forward chaining [61]. RBR uses a given input and finds applicable rules by matching against the rules of the knowledge base. Then, the inference engine will use the obtained rules to generate results using the chosen inference mechanism. In recent years, RBRs have been successfully applied to detect known attacks (e.g. Snort, Suricata, OSSEC), and they were often used as a complementary technique in hybrid detection systems (e.g. Splunk, Elastic search) for the detection of advanced persistent threats and multi-step attacks.

Review. In Medical Cyber Physical Systems, R. Mitchell *et al.* [211] proposed a behavior-rule specification-based technique to detect abnormal patient behaviors and attacks such as data modification, forgery, greyhole and blackhole. The proposed model took a set of behavior rules in input (e.g. (Analgesic Infusion Rate > 0) $\wedge$ (Mode = DEFIBRILLATOR)) and detected if a device's behavior deviated from the expected behavior specified the rule database. In addition, the behaviour rules were transformed in state machine for model checking. In testing phase, the authors simulated vital sign from monitors and evaluated their approach against the aforementioned attacks. As a result, the proposed model achieved a high detection rate of 92.408% and low FPR of 0.666%.

1.4.2.2 Case-based Reasoning

Case-based Reasoning (CBR) [179] is a problem solving paradigm that can adapt old solutions to meet new demands, using old cases to explain/critique new situations, or reasoning from precedents to interpret a new situation or create an equitable solution to a new problem. A general CBR cycle is described by four processes [15]: (a) retrieves the most similar cases, (b) reuses the information and knowledge in that

cases to solve the problem, (c) revises the proposed solution, and (d) retains the parts of this experience likely to be useful for future problem solving.

Review. In Medical Healthcare Systems, D. Malathi *et al.* [87] proposed a CBR-model based on Fuzzy set theory and k-Nearest Neighbor (k-NN) to predict patient anomalies while maintaining the security of patients' health record. The CBR model was used to produce more personalized suggestions and refine the suggestions using patients' treatment success rate to enhance the accuracy of the prediction. The model was combined with Fuzzy k-NN to retrieve the similar cases of the target patients' case record more effectively by estimating the similarity score. Successful target case records are securely stored into the knowledge repository for future reuse. The security of patients' health record is done using Paillier Homomorphic Encryption to prevent from unauthorized user access. During training, the authors selected features such as the gender, age, Total Bilirubin, Direct Bilirubin (DB), Total proteins, Albumin, Ratio of Albumin and Globulin. They trained the model using the Indian Liver Patient dataset from UCI repository. Overall, the proposed model achieved a high ACC of 96.74% and a medium FPR of 16%.

1.4.2.3 Ontology-based Reasoning

An Ontology [332] is referred as the shared understanding of some domains. It is often conceived as a set of entities, relations, functions, axioms, and instances. The World Wide Web Consortium (W3C) created a semantic Ontology Web Language (OWL) for Ontology definition and sharing. An OWL is a Description Logic-based Ontology language compatible with XML for transport layer, Resource Description Framework (RDF) and other extended languages like Semantic Web Rule Language (SWRL) [144] for expression of rules and logics, and Semantic Query-Enhanced Web Rule Language (SQWRL) [242] for OWL queries. Recently, an Ontology-based reasoning (OBR) has been successfully applied in intrusion detection systems, by endowing them with context-awareness and situational-awareness of vulnerabilities, attacks, and network topologies of target environments.

Review. G. Xu *et al.* [333] proposed an OBR-based network security situational awareness (NSSA) model to provide a real-time and holistic view of an IoT network security situation. The NSSA reasoning process is divided into four steps: (a) cap-

turing multiple heterogeneous information (alert, vulnerabilities, context information, network flows) from different embedded IoT sensors, (b) preprocessing heterogeneous network data into an ontology format (OWL), (c) mapping resulting data to the ontology model; CAPEC and CVSS were used for modeling attacks and vulnerabilities respectively, and (d) reasoning on the ontology model to find out anomalies using SWRL and SQWRL.

1.4.2.4 Fuzzy-based Reasoning

A Fuzzy-based Reasoning (FBR) [109] is similar to human reasoning; it uses fuzzy knowledge (knowledge which is vague, uncertain, ambiguous, probabilistic in nature) to generate decision. The FBR system consists of four modules: (a) a knowledge base which contains a set of "IF-THEN" rules, (b) a fuzzification module that transforms input or crisp values into fuzzy sets, (c) an inference engine that applies fuzzy values to the system's rules, and (d) a defuzzification module that outputs the crisp value from a set of fuzzy variables, for example, using a combinatorial function like the center of gravity. As the attacker's behaviour is unpredictable, FBRs may be adapted for intrusion detection systems and this is why many recent work [84, 232, 260] integrated the FBR system in their approach.

Review. In Medical Cyber Physical Systems, multiple medical sensors generate a number of false alarms that could affect the accuracy of sensors. To address this problem, W. Li *et al.* [196] proposed a Medical Fuzzy Alarm Filter based on fuzzy if-then rules to detect anomaly in medical CPS. The authors also mentioned that their alarm filter approach could be also used in intrusion detection to decrease unwanted alarms in a post-preprocessing manner. To realize the fuzzy if-then rules, the authors modified the fuzzy grid partition algorithm in the fuzzy-weka project. In the training phase, the authors trained the model using six features: Heart Rate (HR), Pulse, Respiration Rate (ResR), Blood Pressure (BP), and Oxygen Saturation (SpO2). The model is then applied on three datasets having respectively 533 false alarms and 683 true alarms, 683 false alarms and 1301 true alarms, 1179 false alarms and 523 true alarms. As a result, the proposed model was able to better reduce the alarms in a range from 62% to 83% than existing approaches such as neural network (NN), support vector machine (SVM), decision tree (DT), and Naive Bayes (NB).

N. Naik *et al.* [232] also proposed a dynamic fuzzy rule interpolation approach based on interpolated rules, GA algorithms and the Snort tool to detect attacks in the network traffic. The GA model was responsible for finding the best clusters (solutions) without assigning a predetermined number of emerging clusters. At the end of the clustering process, only those clusters (solutions) containing adequate interpolated rules are selected for the subsequent rule promotion process. After several tests, the proposed model considerably reduced false alarms.

1.4.3 Muli-Agent based detection techniques

An agent [268] is an autonomous entity able to perceive its environment through sensors and to act upon that environment through effectors. Like humans, agents possess a body of factual knowledge (ML, KBS, heuristics), belief, goal that give them the ability to adapt to their environment, reason and act (see Fig. 1.12). There are various kinds of agents including reactive agents that simply convert its sensory inputs into actions without maintaining its internal state (e.g. memory based on rules) and deliberative agents that maintain its internal state (e.g. memory based on predictive or complex models) and can predict the effects of its actions. A Multi-Agent System (MAS) [306] is a society or network of agents which can work independently or cooperatively to achieve their goal together. The goal being, for example, to find solutions to problems. In computer security, MASs are widely used in distributed/centralized network infrastructures to perform various tasks such as information gathering, intrusion monitoring, fault monitoring in CPS systems, health supervisory and management. In intrusion detection systems, well-known multi-agent detection techniques includes centralized multi-agent-based detection, collaborative multi-agent-based detection, and cooperative multi-agent-based detection.

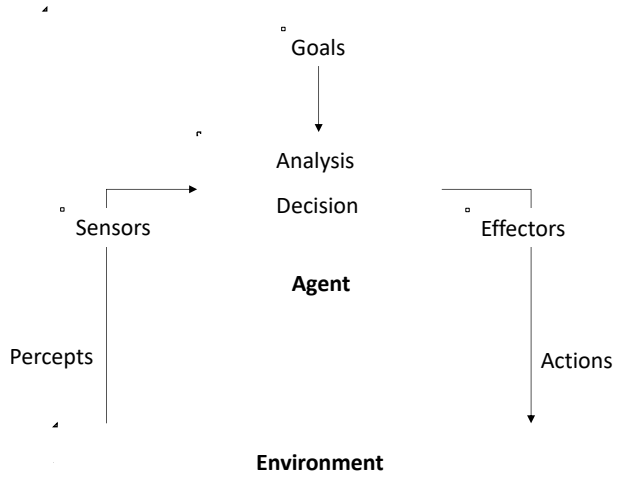


Figure 1.12 – An agent

1.4.3.1 Centralized Multi-Agent-based detection

Centralized Multi-Agent-based detection uses a set of local agents and coordinator agents for the detection of complex attacks such as DDoS, Botnets and Spam campaigns. Each local agent can perform various tasks such as gathering network data from end-points and network nodes, local filtering or pre-processing of network data, local analysis of data sources generated by web servers/softwares/operating systems, alert triggering and sending of information to a remote coordinator agent. In the central C&C server, coordinator agents manage the received information, and make different operations including pre-processing of heterogeneous/homogeneous events, aggregation/correlation/fusion of received events, deep analytics of received events, and reporting of intrusions or system's behaviours.

Review. In Smart Grids, T.R.B. Kushal *et al.* [177] proposed a strategy to mitigate the effects of FDI attacks using battery to actively reduce load curtailment and a MAS approach. The MAS approach consists of agent nodes that check commands from the central energy management system and identify malicious commands from malicious nodes using a heuristic model. The authors evaluated their model through a risk analysis mode and the results shown that the model is accurate in mitigating FDI attacks.

R. Kwon *et al.* [174] proposed a centralized network intrusion forecasting and de-

tection system to predict and detect DDoS attack volume in their university. The proposed system relies on the Honeypot network system (Honeynet) where honeypot sensors collect various data (audit logs, network traffic) and send it to C&C servers for forecasting and detection. In C&C server, both correlation and regression statistical approaches were used for deep data analytics. The approaches had to take into account two factors: (a) external factors such as the number of C&C servers and the size of malicious bot agents, and (b) internal factors such as the security level of the resources and their values. Through experiments (over 8 months), the proposed model predicted DDoS attack volume and helped network operators to decide when installing IPSs or not.

1.4.3.2 Collaborative Multi-Agent-based detection

Collaborative Multi-Agent-based detection is an approach where agents collaborate (mutual trust) i.e. share knowledge and experience together, and then, dynamically update and improve their performance to achieve their goal. The goal can be the information gathering, pre-processing and analysis of network data, negotiating (P2P communication) on intrusion results whether results are not accurate, ejecting malicious nodes in ad-hoc networks and alerting when an attack occurs. This approach is particularly useful for self-configuring infrastructure-less dynamic wireless networks such as UAVNETs, MANETs or VANETs.

Review. In WSNs, Wenjuan Li *et al.* [195] proposed a trust-based collaborative IDS which allows IDS agents to collect network data on its node and learn experience of others using their detection sensibilities and a trust model between them. Thus, IDS agents improve their detection accuracy using learned knowledge, based on machine learning techniques. In the testing phase, the authors trained the model using a simulated WSN traffic and compared the performance to some existing supervised classifiers. They achieved a good performance and enhanced the detection accuracy of malicious nodes.

Recently, C.J. Fung *et al.* [106] proposed a trust-based collaborative IDS framework (FACID) that consists of a trust management model and a cooperative model.

The trust management model allows each agent to aggregate feedback (trust messages) received from its acquaintances (i.e. other IDSs) using a weighted majority algorithm, and to evaluate the trustworthiness of others based on its own experience with them. The cooperative model connects each IDS agent together using an encrypted P2P communication, so that they can securely communicate and cooperate with each other to achieve better intrusion detectability. The simulations showed that the proposed approach was more accurate, cost efficient and robust than other heuristic methods.

1.4.3.3 Cooperative Multi-Agent-based detection

Cooperative Multi-Agent-based detection assumes that agents cooperate (mutual respect) together through their interaction, to achieve a join goal. For example, the goal can be to detect an attack or proactively defend a given system.

Review. In Smart Grids, M. S. Rahman *et al.* [267] developed a distributed agent-based cooperative framework that used a group of agent-based IDSs to analyze network monitoring logs, relay status logs, synchrophasor data, and to detect faults, cyber attacks in each substation of a power system. In the testing phase, the authors simulated their model using MATLAB and used Java Agent Development Framework (JADE) to develop three agents: a root node agent for the monitoring of relay activities, a protected relay agent for the capture of protection parameters (e.g. status of relays) and a base station agent for attack detection. Thus, the authors were able to measure the effect of the proposed approach that effectively distinguished attacks from faults.

1.4.4 Hybrid detection techniques

Most recent work used a hybrid detection as they are more accurate to prevent and detect common threats. But, they are often expensive in terms of performance due to the integration of various detection mechanisms. In the literature, hybrid detection combined one or more topics like classification, evolutionary computing, deep learning, multi-agent systems, reinforcement learning, knowledge base engineering,

active learning and regression. Some hybrid methods are presented in this section, precisely, Multi Modal Deep Learning (DL^n), SVM and ELM (Extreme Learning Machine), Clustering and ELM, Fuzzy C-Means and ANN, Population-based Incremental Learning (PBIL) and AIS.

1.4.4.1 DL^n

T. Kim *et al.* [175] proposed a multimodal deep Learning approach for Android Malware Detection. The multimodal DNN consisted of 5 DNNs, each associated to a feature vector. Initially, DNNs are not connected to each other. Only the last layers of the initial networks are connected to the merging layer that is the first layer of the final DNN. The authors considered 5 kind of Android features to build training feature vectors such as String feature, method API feature, method opcode feature, shared library function opcode feature, and permission/component/environmental feature. These features are extracted from Smali which is an assembler/disassembler for the dex format and the instruction sequences of the disassembled code of .so files. In the testing phase, the proposed model was implemented using the Keras library and the Malgenome dataset. Overall, the proposed model achieved a high ACC of 98%.

1.4.4.2 SVM and ELM

W. Laftah *et al.* [18] proposed a hybrid approach (SVM-ELM) that combines SVM and Extreme learning machines (ELM) [151] for the classification of known and unknown attacks. In the training phase, the K-mean algorithm is used to transform KDD cup 1999 dataset into small datasets representing the whole dataset. Then, resulting datasets were used to train the SVM-ELM model and to improve the training time as well as the detection accuracy. Simulations showed that the proposed solution achieved an overall ACC of 95.75% on the entire improved dataset.

1.4.4.3 Clustering and ELM

S. Roshan *et al.* [266] developed an adaptive and real-time hybrid IDS that groups clustering and ELM techniques to accurately classify known and unknown intrusions. The model consists of three components: (a) a clustering manager that maps training

data into clusters (a class of traffic), (b) a decision maker that evaluates the clustering decisions and provides correction proposals, and (c) a update manager that updates the clustering model using new information from a human expert. In the experimental phase, the authors trained and evaluated their approach using NSL-KDD. The proposed model gave good results i.e. a high DR of 84% and a low FPR of 3%.

1.4.4.4 Fuzzy C-Means and ANN

In cloud computing, N. Pandeeswari *et al.* [248] proposed a hybrid IDS that used Fuzzy C-means (FCM) clustering [27] and ANN to efficiently detect attacks. The model is implemented in the middleware i.e. the virtual machine monitor (VMM) layer where it analyzes the network traffic and automatically captures new attack patterns. The authors trained and tested the model using the KDD cup 1999 dataset. Afterwards, they compared results with existing methods like Naive classifier and ANN. As a result, the authors achieved better performance than other techniques i.e. a high ACC over 95%, and a relative low FPR less than 6%.

1.4.4.5 DAE and SVM

M. Al-Qatf *et al.* [13] proposed a hybrid method based on Sparse autoencoders and SVMs for network intrusion detection. In the pre-training phase, Sparse autoencoders were used to reconstruct low-dimensional features from raw data. Next, the resulting features are fed into the SVM algorithm to improve its intrusion detection accuracy. In the testing phase, the authors evaluated the model using the NSL-KDD dataset and also analyzed the impact of low-dimensional features and sparsity parameter on the SVM classifier. As a result, the proposed model achieved better performance than a single SVM, with a high ACC of 84.96% and DR of 76.57% for testing datasets, and a high ACC of 99.416% and DR of 99.291% for training datasets.

1.4.4.6 Population-based Incremental Learning and AIS

M. Chen *et al.* [58] combined AIS approach and Population-based incremental learning (PBIL) [22] for cyber attack detection. PBIL groups GA and Simple Competitive Learning (SCL) methods [276]. The hybrid approach has three phases: (a)

network traffic capture and pre-processing or network feature extraction, (b) computation of affinity between antibodies and antigens, creation of new antibodies and (c) detection of attacks. In the experimental phase, KDD cup 1999 and ACA (Australia Credit Approval) datasets were used to train and evaluate the proposed model. As a result, the proposed solution outperformed existing evolutionary approaches with a high ACC of 97.03%.

1.5 Review and Evaluation of Event Stream Processing methods

Many things are increasingly connected to the Internet and generate tremendous heterogeneous data streams in real time. Precisely, more than 2.5 quintillion bytes of data are generated every day and this number is growing exponentially [8]. Thus, some intrusion detection techniques previously described may not be adapted. To deal with big event streams, Event Stream Processing (ESP) methods are advanced approaches that take advantage of large datasets or event streams to provide real-time and proactive insights for intrusion detection. The nature of events depends of the environment where they were produced, see Table 1.5.

8. https://www.ibm.com/analytics/hadoop/big-data-analytics

Table 1.5 – Examples of events in different application domains

Environments		Events
Industrial and Control Systems (ICSs)	Power systems (Smart Grids)	electric current (Voltage, Intensity, Power, Energy), temperature
	Water Treatment Plant	chemical (hydrochloric acid, sodium hypochlorite), temperature, electric current
	Nuclear Power Plant	temperature, electric current
Intelligent Transportation systems (ITSs)	Smart vehicles	geo-coordinates, fuel level
	Unmanned Aerial Vehicles (UAVs)	battery level, geo-coordinates, urban images or videos
	Smart trains	geo-coordinates, fuel level
Health care systems	Wearable peacemakers	Heart beats
	Blood Pressure Monitors	Blood pressures (diabetes, hepatitis, alcohol, stress)
	Implantable cardioverter-Defibrillators	Heart beats or Heart Rhythms
Building systems	Smart homes	Temperature, pressure, human activities, electricity current, light brightness, Door open/close
	Smart cities	Weather, seismic, urban images or videos
Banking systems	Automated Teller Machines (ATMs)	Money transactions (withdrawal, deposit, transfer, inquiry)
	Electronic Funds Transfer (EFT) systems	

Events can be natural (physical) and non-natural. Natural events represent sensible nature-based raw data generated by cyber-physical systems like Industrial and Control Systems (e.g. temperature, electrical energy, chemical compounds and physical location), animal and human bodies (e.g. blood pressure, heart beats), and the natural ecosystem (e.g. weather, seismic or human activities). Non-natural events are artificial or man-made data such as numerical/digital signals (e.g. current inten-

sity/voltage time series), heart beat or blood pressure time series, network packet flows, and log file events generated by web servers, operating systems, applications or services.

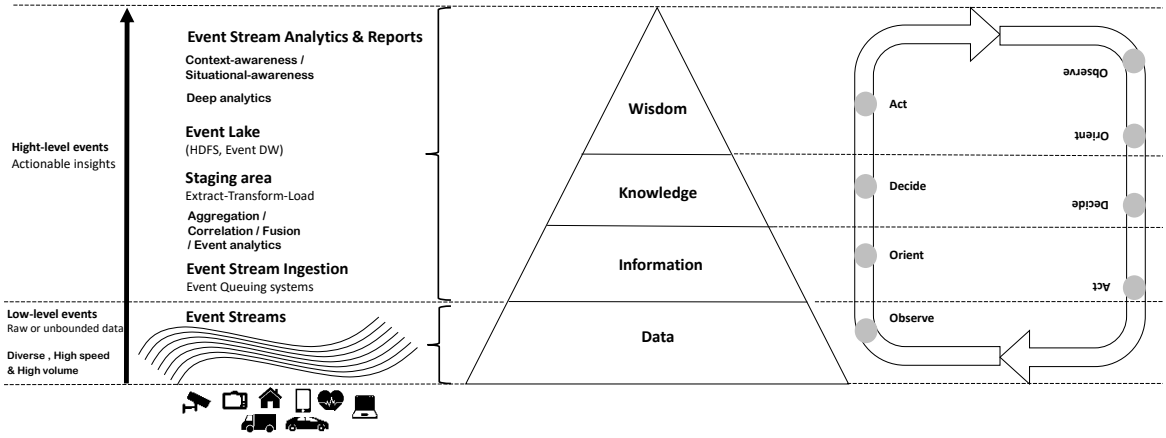


Figure 1.13 – Event Stream Processing Chain

Events can be classified according to different levels of abstraction (see Fig. 1.13). Generally, two powerful models are used together to better describe the hierarchical levels: Data-Information-Knowledge-Wisdom (DIKW) [272] pyramid and Observe-Orient-Decide-Action (OODA) loop [47].

a) Data Level. A huge number of diverse security events are captured in real-time from a plethora of end-points and embedded devices by hardware and software event collectors such as honeypots, IDSs, IPSs, anti-virus, firewalls, and custom event collectors made by cybersecurity companies. Event collectors observe the target environment and push-out remotely raw or unbounded events for in-memory preprocessing, analysis and storage. At this level, the raw events, also called low-level events, have not yet been processed for use.

b) Information Level. Collected security event streams are transformed into actionable events (i.e. high-level events) to get orientations for the decision making. The process cycle of security events is managed by a Security Information and Event Management (SIEM) infrastructure that provides real-time analysis of security event streams and supports various cloud technologies (e.g. OpenStack). During the pro-

cess, security event streams are fed into the staging area or landing zone responsible of ETL (Extract-Transform-Load) operations and temporal storage in data marts and warehouses.

Upon starting, event streams are ingested in Event Queuing Systems (EQSs). EQSs are high-speed data ingestors (e.g. Apache Kafka) that manage and store efficiently events in a queue. They enable real-time processing of event streams by offering a straightforward mechanism for managing over-information in the queue and avoiding lock or concurrency problems. EQSs have been proposed to overcome some limitations of the Event Batch Systems (EBSs) which compute results that are derived from event stream batches or groups (batch processing). This approach is not often suitable for stream analysis because it takes long waiting time (high latency) and makes analysis more complex as the size of batches can be very large.

Ingested event stream queues are fine-tuned using Complex Event Processing (CEP). A CEP allows complex computations on event frames using Map Reduce [81], Machine Learning and Data Fusion [139]. CEPs are often assimilated to ESPs while they are just a subset of ESPs. CEPs and ESPs differ in terms of processing engines [203] due to many reasons. First of all, CEP engines support a SQL-like query language while there is often need to write scripts or codes in ESPs. Secondly, ESP engines are distributed and parallel natively opposite to CEP engines which are more centralized. Most known tools such as Microsoft Azure Stream Analytics, Apache Spark and Kafka support real-time CEP for event analytics.

After CEP computations, high-level event streams are then stored in big long-term databases (event lake) such as Hadoop Distributed File System (HDFS) and Relational Database Management System (RDBMS). Offline SQL-based stream processors can be used for querying live security event streams (e.g. Apache Hive, Apache Pig, Apache HBase).

c) Knowledge Level. High-level event streams are fetched from the event lake, and refined using online or offline deep analytics to gain new security event insights for more accurate decisions. Deep analytics integrate advanced ESP approaches such as deep learning, text analytics, predictive analytics, data mining, statistics and knowledge-based reasoning. In the next subsections, we will describe those approaches and related recent work.

Unlike deep analytics, data streams can be processed at the edges of the network. This concept was recently introduced by Cisco and it is called Fog analytics. The fog is a cloud close to the ground [45]. Fog analytics [45, 183] leverage the extension of cloud capabilities to the edge of the network, precisely at (or closer to) the source of the event to perform real-time processing locally, and just send security event back to the cloud, rather than moving massive amounts of raw events.

Generally, deep analytics are more centralized while fog analytics are decentralized to the edge of the network. To be more accurate, deep analytics often take into account contextual information to better understand the whole cybersecurity ecosystem (context awareness) and the current situation of cyber threats occurring on the monitored system (situational awareness). Contextual information includes platforms (CPE), vulnerabilities (CVE), software weaknesses (CWE), attack classes (CAPEC), network topology and all other system-related information. It is often organized into an intuitive knowledge-based languages like Ontologies. The combination of deep analytics and context/situational awarenesses help SIEM system to reason and take accurate decisions.

d) Wisdom Level. The SIEM system can reason based upon its knowledge-base and act with/without human intervention. In this level, cyber threat information is structured in attack languages such as STIX and routed between organizations using the MITRE's Trusted Automated eXchange of Indicator Information (TAXII) mechanism. It is a standard that allows sharing cyber threat information using various models such as Producer/Consumer, Peer-to-Peer (P2P) and Source/Subscriber.

In recent papers, various ESP techniques have been applied for preprocessing, processing and deep processing of event security streams. In Table 1.6, we have just reviewed some of them in the following topics: aggregation, fusion, correlation, knowledge-based reasoning and deep analytics. We also use the notation R which denotes the reduction rate of false alarms.

Table 1.6 – Evaluation of recent event processing techniques

Papers	Domain	Architect.	LCT	Source	Technique	Parent Tech.	Datasets	Results
Q. Chen *et al.* [63]	ITS	VANET	✗	event stream	AnRAD	Deep Analytics	Testbed	Good
G. Xu *et al.* [333]	Big Data	✗	✗	event stream	OBR	Knowledge-based	Simulation	Good
W. Wang *et al.* [331]	Big Data	✗	✗	Traffic	Tensor DL	Data Fusion	STL+CUAVE	A:84.96%
J. Abawajy *et al.* [12]	Big Data	✗	✗	Traffic	–	Data Fusion	–	–
S. Noel *et al.* [236]	Big Data	✗	✗	security events	CyGraph	Deep Analytics	–	–
L. Yang *et al.* [339]	Big Data	✗	✗	event stream	HBST+DDF	Aggregation	Testbed	D:90%
J. M. Vidal *et al.* [321]	Big Data	✗	✗	alarm events		Correlation	Testbed	Good
R. Shittu *et al.* [291]	Big Data	✗	✗	alarm events	–	Correlation	Testbed	R:97%
S.N. Narayanan *et al.* [239]	ITS	VANET	✗	event stream	OBR	Knowledge-based	Simulation	Good
A. Tuor *et al.* [314]	Big Data	✗	✗	event stream	DNN	Deep Analytics	CERT	D:95.53%
V. Shah *et al.* [278]	Big Data	✗	✗	alarm events	Dempster-Shafer	Data Fusion	KDDcup99	A:73.32%; F:0.73%
S. Samarah *et al.* [277]	SmartHomes	✗	✗	Health data	NB+aggreg.	Aggregation	Kyoto	A:98%
Z. Tian *et al.* [309]	SmartCampuses	✗	✗	Traffic	–	–	Correlation	–

1.5.1 Aggregation-based

In Large-scale infrastructures, a big amount of events is generated at any local instances of sensors. More often, gathering all events and trying to look at each feature is humanly unfeasible. In addition, centralizing all gathered events for further processing in C&C central is relatively time-consuming and expensive [7]. Thus, the aggregation or summarization process appears to be one of solutions that compactly reduces redundant events. However, aggregation remains a complex process (can be NP-hard) that requires hardware optimization and additional hardwares, processing optimization and task scheduling [317]. In recent work, aggregation has been differently applied in distributed mode i.e. at the edges of the network, locally or closer to the source of the event and in centralized mode i.e. at the C&C network node.

Theory. An aggregation takes multiple inputs and outputs a summary from these inputs using an aggregation function. Let $(x_1, ..., x_m) \in \mathbb{I}^m$ be a finite data frame in the data streams and $y \in \mathbb{I}$ an output data. An aggregation function [79, 108] is defined by $F : \mathbb{I}^m \to \mathbb{I}$, such that $y = F(x_1, ..., x_m)$ and satisfies several properties

such as,

$$
\begin{aligned}
&(1)\ F(x) = x \\
&(2)\ F(x, ..., x) = x \\
&(3)\ F(x_1, ..., e, ..., x_m) = F(x_1, ..., x_m) \\
&(4)\ F(x_{\sigma(1)}, ..., x_{\sigma(m)}) = F(x_1, ..., x_m) \\
&(5)\ F(x_1, ..., x_m) \geq F(x'_1, ..., x'_m) \ \text{ if } \ x_i \geq x'_i \\
&(6)\ F(x_1, ..., x_m) = F(F(x_1, .., x_j), x_{j+1}, ..., x_m) \\
&\qquad\qquad = F(x_1, ..., x_j, F(x_{j+1}, ..., x_m)) \\
&(7)\ F(\alpha x_1 + \beta, .., \alpha x_m + \beta) = \alpha F(x_1, .., x_m) + \beta
\end{aligned}
$$

where e is the neutral element, α and β are two reals. Simple aggregation functions are MIN, MAX, SUM, PRODUCT, MEDIAN, AMEAN (arithmetic mean) and WMEAN (weighted mean). Input data streams are often normalized and scaled before processing using functions such as standardization (ST), robust standardization (RST), and normalization (NORM).

$$\mathsf{ST}(x) = \frac{(x - \mathsf{AMEAN}(x))}{\mathsf{SD}(x)}$$

$$\mathsf{RST}(x) = \frac{(x - \mathsf{MEDIAN}(x))}{\mathsf{MAD}(x)}$$

$$\mathsf{NORM}(x) = \frac{(x - \mathsf{MIN}(x))}{\mathsf{MAX}(x) - \mathsf{MIN}(x)}$$

where $\mathsf{SD}(x)$ is the standard deviation and $\mathsf{MAD}(x)$ is the median absolute deviation. They are of the form,

$$\mathsf{SD}(x) = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \mathsf{AMEAN}(x))^2}$$

$$\mathsf{MAD}(x) = 1.4828.\mathsf{MEDIAN}(|x - \mathsf{MEDIAN}(x)|)$$

More complex aggregations functions include clustering and neural networks. Neural network aggregation functions are of family of hierarchical aggregation functions. A hierarchy of fusion functions is defined by the tuple $\mathcal{H} = \langle l, m, F \rangle$, where $l \in \mathbb{N}$ is the number of layers, $m = (m_1, ..., m_l) \in \mathbb{N}^l$, where $m_i = n$ is the number of aggregation functions in layer $i \in \{1, ..., l\}$, $m_1 = n$ the number of inputs, $m_l = 1$ is the number of output, and $F = (F_j^{(i)})_{i \in I, j \in J}$ is a sequence of aggregation functions, such that $F_j^{(i)} : \mathbb{I}^{m_{i-1}} \to \mathbb{I}$, $I = \{1, ..., l\}$ and $J = \{1, ..., m_i\}$. In the case of feedforward neural network aggregation, given an input example $(x_1, ..., x_{m_{i-1}})$, $F_j^{(i)}$ is given by

$$F_j^{(i)}(x_1, ..., x_{m_{i-1}}) = f\left(\sum_{k=1}^{m_{i-1}} w_{kj}^{(i)} x_k + w_{0j}^{(i)})\right)$$

where f is an activation function, $w_{kj}^{(i)}$ are weights of the i-th layer and $w_{0j}^{(i)}$ are biases of the i-th layer.

Review. In Smart Homes, S. Samarah *et al.* [277] combined the spatiotemporal mining approach, k-anonymity, micro-aggregation to efficiently recognizes human activities and preserve health data from inference-based privacy attacks. The proposed model collected health data from smart homes and weighted the readings from network sensors using a probabilistic feature extraction technique [277]. The data set consists of a set of events in which each event is associated with a date, time, sensor id, and sensor status (Open-Close, On-Off). Profiles was extracted from dataset and converted into a set of feature vectors. The collection of vectors is fed into a Naive Bayes classifier for activity profiling. A k-anonymity model uses a micro-aggregation to divide data into many equivalent classes such that the values of an identification attribute of any record in the dataset are similar to at least k-1 records. The authors evaluated the model using Kyoto, Milan and Tulum datasets. Overall, the proposed model achieved a high ACC of 98% for Kyoto, 91% for Milan, and 92% for Tulum.

In IoT, L. Yang *et al.* [339] proposed a hybrid approach based on a divided difference filtering (DDF) and a hierarchical Bayesian Spatial-Temporal (HBST) model

[324] for false data injection (FDI) attack detection, when the aggregation process is being compromised by malicious nodes that inject false aggregated data. The authors modeled the attack scenario using an adversarial game model between FDI attacks and defenders (based on their schemes). Next, they evaluated their approach and achieved a high DR (greater than 90%) and a low FPR.

1.5.2 Fusion-based

Data Fusion [140] is analogous to the ongoing cognitive process used by humans to integrate data continually from their senses to make inferences about the external world. It fuses heterogeneous security events from multiple and various sensors into a synthetic, accurate, and consistent representation. However, the fusion process depends of computational resources and time delays when transferring information between the different sources [54]. Security event streams can be fused together using various data fusion approaches such as data mining-based data fusion (i.e. K-Nearest Neighbor (KNN) [34], probabilistic data association (PDA) [54]), stage-based data fusion [348], feature-level-based data fusion [348] like DNN and feature concatenation, semantic meaning-based data fusion [348] like similarity and co-training, Joint Directors of Laboratories (JDL) and Demptser-Shafer Theory-based Decision Fusion [54].

Theory. Data fusion is a little bit similar to data aggregation. Data aggregation is essentially used to provide information in a synthetic form (summary of data) while data fusion integrates multiple heterogeneous data, leverages some aggregation techniques and extracts actionable insights from this data [108]. In the literature, the most widely-used data fusion model is the JDL data fusion proposed by the Joint Directors of Laboratories (JDL) Group and the American Department of Defense (DoD).

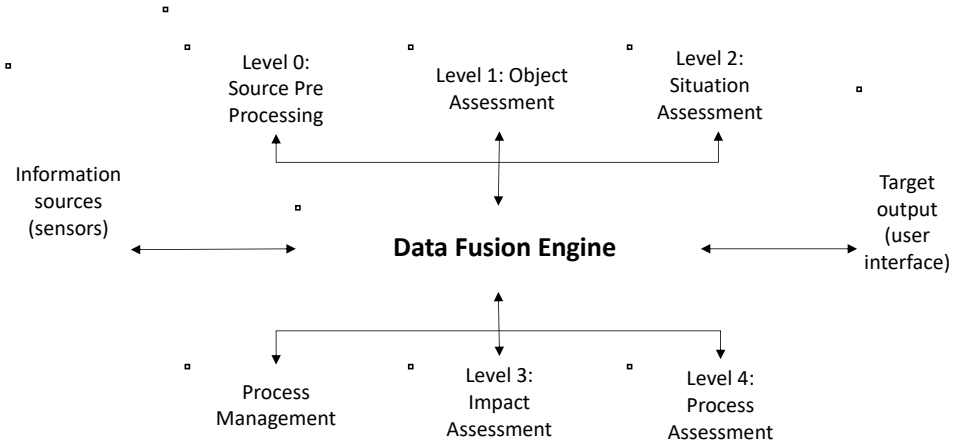


Figure 1.14 – JDL Data Fusion Model

The JDL data fusion model has five processing levels [139]: source preprocessing that preprocesses low-level data of sensors; object refinement that fine-tunes data into valuable information using data fusion techniques like association, correlation, clustering and state estimation; situation assessment that identifies the likely situations from the observed events and builds relationships between them; impact assessment that evaluates the current situation to identify possible risks and vulnerabilities; and process refinement that improves the previous steps to achieve efficient resource management (see Fig. 1.14). In the JDL levels, Distributed Joint Probabilistic Data Association (JPDA-D) [56] and Dempster-Shafer Inference-based Decision Fusion (DSI) [228] are respectively used to track multiple targets in cluttered environments and to make decisions, based on the knowledge of the perceived situation from sensors in the data fusion domain. JPDA-D estimates state of the target, given two sensors as follows,

$$\mathbb{E}[x|Z^1, Z^2] = \sum_{j=0}^{m_1} \sum_{k=0}^{m_2} \mathbb{E}[x|\chi_j^1, \chi_k^2, Z^1, Z^2] P(\chi_j^1, \chi_k^2|Z^1, Z^2)$$

where Z^1, Z^2 are the set of accumulative data, m_1 and m_2 are the last set of measurements of sensor 1 and 2, χ is the association hypothesis, χ^1 and χ^2 are the joint hypotheses involving all measurements and objectives, $\mathbb{E}[x|\chi_j^1, \chi_l^2, Z^1, Z^2]$ the expected estimated state from previous associations. The data association probabil-

ity $P(\chi_j^1, \chi_k^2|Z^1, Z^2)$ is defined by

$$P(\chi_j^1, \chi_k^2|Z^1, Z^2) = \sum_{\chi^1} \sum_{\chi^2} P(\chi^1, \chi^2|Z^1, Z^2)\hat{\omega}_j^1(\chi^1)\hat{\omega}_k^2(\chi^2)$$

with

$$P(\chi^1, \chi^2|Z^1, Z^2) = \frac{1}{c}P(\chi^1|Z^1)P(\chi^2|Z^2)\gamma(\chi^1, \chi^2)$$

where $\hat{\omega}_j^1(\chi^1), \hat{\omega}_j^2(\chi^2)$ are the binary indicators of the measurement-target association. $\gamma(\chi^1, \chi^2)$ reflects the localization influence of the current measurements in the joint hypotheses χ^1, χ^2 and it is based on the correlation of the individual hypothesis.

In DSI, the Dempster-Shafer relies on the mass function, belief and plausible functions, and connection with rough sets. Let Ω be a frame discernment or possible states of the world, a mass function $m : 2^\Omega \to [0, 1]$ is defined such that $m(\varnothing) = 0$ and $\sum_{H\subseteq\Omega} m(H) = 1$, where H is an hypothesis. A mass is induced by a source $\langle \mathcal{S}, 2^\mathcal{S}, P, \Gamma \rangle$, where $\mathcal{S}$ is a finite set of interpretations of the evidence, P is a probability measure on $(\mathcal{S}, 2^\mathcal{S})$, and Γ is a multi-valued mapping from $\mathcal{S}$ to $2^\mathcal{S}$. For any hypothesis $H \subseteq \Omega$, the incomplete beliefs in H is defined by a belief function $\mathsf{Bel} : 2^\Omega \to [0, 1]$, such that

$$\mathsf{Bel}(H) = P(s \in \mathcal{S}|\Gamma(s) \subseteq H) = \sum_{X\subseteq H} m(X)$$

The plausibility of each hypothesis H is given by the function $\mathsf{Plau} : 2^\Omega \to [0, 1]$, such that

$$\mathsf{Plau}(H) = 1 - \mathsf{Bel}(\neg H) = \sum_{X\cap H=\varnothing} m(X)$$

The confidence interval $[\mathsf{Bel}(H), \mathsf{Plau}(H)]$ defines the true belief in hypothesis H.

Review. In IoT Big Data, W. Wang *et al.* [331] proposed a tensor deep learning model (TDL) that composes multiple deep learning techniques using tensors to realm processing big heterogeneous data. Tensors are used to model the complexity of multisource heterogeneous data and extend the vector space data to the tensor space. The conventional back-propagation algorithm was extended into a high-order back-propagation algorithm to transform data from the linear space into multiple linear space. In the testing phase, the proposed model was trained and tested using STL-10

and CUAVE datasets. As a result, the proposed model achieved a higher recognition accuracy than the stacked auto encoder and the multimodal deep learning model.

In Big Data, J. Abawajy *et al.* [12] proposed an iterative classifier fusion system for Android Malware detection. The system iteratively applies classifiers in fusion with new iterative feature selection (IFS) procedure. The model achieved better performance when combining the ICFS procedure using LibSVM with polynomial kernel with Multilayer Perceptron and NBtree classifier and applying IFS feature selection based on Wrapper Subset Evaluator with Particle Swarm Optimization.

V. Shah *et al.* [278] proposed a fusion-based distributed intrusion detection system that used an improved Dempster Shafer rule based on Alert-to-mass conversion. The Alert-to-mass conversion transforms real-time heterogeneous alert events into mass or weight values. The authors modified the Dempster Shafer Theory using a combination function of these mass values. In the testing phase, they evaluated their approach using the KDD cup 1999 dataset and achieved a good performance i.e. a low FPR of 0.73% and a high ACC of 73.32%.

1.5.3 Correlation-based

Event correlation relates different events across observation time and space to identify patterns, their relationships and causalities. The observation time represents the window or interval time where events occurred. The observation space takes into account the spacial variation of other features of events in the time. Event correlation is integrated in the SIEM module where it performs various operations [112, 329]: (a) *compression* replaces multiple identical events by a single event; (b) *time-based correlation* correlates events in the time as they occur (e.g. event X follows event Y within 100 milliseconds); (c) *logic operations* connects events with Boolean operator (or, and, not); (d) *counting* reports a specified number of similar events as one; and (e) *clustering* identifies similar events and generates new events from the patterns of clusters. In recent papers, various event correlation techniques were used or combined to detect zero-day attacks including rule-based correlation, case-based reasoning correlation, neural network-based correlation and fuzzy-based correlation.

Review. In Smart Campuses, Z. Tian *et al.* [309] proposed a real-time correlation

of host-level events using evidence graphs and a cyber range service (CRS) method to detect attack evasions. The evidence graphs allow an effective evidence presentation and automated reasoning. A C2RS supports an out-of-band data capturing mechanism for greater attack resistance utilizing virtual machine introspection (VMI) technology. The VMI technology process allows monitoring virtual machines at the hypervisor layer i.e. exposing/exploiting VM runtime state. A C2RS uses cloud controllers such as OpenStack and VMWare vSphere as network management configuration tools. The authors extracted from Virtual Machine, features such as process list, process maps, open ports, open files, library list, and netstats. The evidence feature vectors are stored in the backend database. Then, the correlation engine accessed features and performed evidence analysis and graph reconstruction of the invasion attacks. Through experiments, the proposed approach achieved low-latency detection and was able to reconstruction host attack scenarios (e.g. useradd/groupadd, executes vbs file, write webshells, delete itself).

Recently, J. M. Vidal *et al.* [321] proposed a correlation framework that analyzes, classifies and prioritizes alerts generated by multiple sensors based on payload analysis. The processing engine has two layers. The first layer makes a fast inspection of potential threats in the gathered alerts. The second layer performs deep analysis of alert flows by reconstructing attack scenarios and giving an overview of potential threats. The authors evaluated their approach using a real-time traffic from a test bed and reduce considerably the FPR.

R. Shittu *et al.* [291] developed a correlation-based hybrid model, called *A Comprehensive System for Analyzing Intrusion Alerts* (ACSAnIA), that integrates correlation knowledge (contextual information) for alert clustering and an anomaly detection based on prioritisation metrics. The authors tested their approach using 2012 cyber range experiment carried out by the British Telecom Security Pratice Team. As a result, the proposed model successfully reduced false-positives by 97%.

1.5.4 Knowledge-based

A Knowledge-based Reasoning (KBR) is an approach that uses the background knowledge about events and their relations through ontologies for inferring relations

between events i.e. reasoning on their type hierarchy or temporal/spacial relationships [224]. The knowledge-base contains event meta-data and contextual information related to events or other resources of the system. This knowledge-base helps event analysis to be aware of the context and current situation of the system in order to take accurate decisions. The event analysis engine accesses to the knowledge-base by performing online or offline event query rules (e.g. SQWRL, SPARQL).

Review. G. Xu *et al.* [333] proposed a network security situation awareness (NSSA) model that integrates a semantic ontology about network security entities (netflow, attack, vulnerability, context, alert, sensor), and user-defined rules based on OWL/SWRL. The NSSA model allows providing a holistic view and situational awareness of the security state of the entire network. It has three main levels: (a) security situation perception level collects information from tremendous multi-source heterogeneous data and translates it into comprehensive formats; (b) situation evaluation level identifies the security events and analyzes their relationships among these events to obtain the security situation of the monitored network; and (c) situation prediction predicts the change trend of the network situation in the future. To show the effectiveness of their approach, the authors built and tested some scenarios using Protege for DDoS detection, worm detection and vulnerability assessment. As a result, the model showed a good performance in all test cases.

In VANETs, S.N. Narayanan *et al.* [239] developed a semantic-based context mining system for context-awareness of the attack surface (internal or external hardwares, on-Vehicle devices) and vulnerabilities occurring in vehicular IoT. The system consists of four layers: (a) a Local Context Detection (LCD) layer collects streaming data from a vehicle's controller area network (CAN) bus, extracts contextual information (e.g. high speed, normal speed) and associates it to their respective ontology entities; (b) a Cross Component Context Inferencing Engine (C3IE) layer aggregates contextual information from LCD to infer overall state of the system using SWRL/SPARQL reasoners; and (c) a Rule Mining Engine extracts knowledge from historical data to support the inferencing process. The authors evaluated their approach by representing different abnormal scenarios in the Protege tool and testing whether the proposed model can detect these scenarios.

1.5.5 Deep analytics

Deep Analytics is a multidimensional approach that uses mathematics, business models and intelligent methods (e.g. machine learning, data mining) to find meaningful patterns in depth, or predict behaviours of a system. Recently, different deep analytics were applied in the intrusion detection such as descriptive analytics that create a summary of historical events for data preparation, diagnostic analytics that look events in depth to figure out their behaviours and causalities, predictive analytics that give trends about the system behaviour using predictive algorithms and past events (e.g. regression, SVM), and prescriptive analytics that make real-time decision using predictive analytics and knowledge-based reasoning for cyber threat detection.

Review. In VANETs, Q. Chen *et al.* [63] proposed a neuromorphic anomaly recognition and detection (AnRAD) framework that learns an efficient self-structured confabulation network (corrupted neural memory) using streaming data, and processes diversified incoming concurrent data streams in parallel using a detection algorithm based on the graphical processing unit (GPU) and the Xeon Phi coprocessor. The authors tested the framework using a vehicular network testbed and the framework was able to monitor more than 16 000 vehicles (event streams) and their interactions in less than 0.2ms for one testing subject.

During the MITRE's Cyber Situational Awareness Solution project, S. Noel *et al.* [236] proposed a unified graph-based analytic model called CyGraph which has four main tasks: (a) capturing attack, vulnerability and real-time security events; (b) building a predictive model of possible attack paths and critical vulnerabilities; (c) correlating security events or intrusion alerts to known attack and vulnerability paths, and then (d) providing a situational awareness of vulnerabilities and attacks occurring on the monitored network. CyGraph supports a query language (CyQL) for expressing graph patterns by querying cyber security assets with an interactive visualization of query results.

A. Tuor *et al.* [314] proposed an online unsupervised DNN approach to filter system log data streams and detect network activity from system logs in real-time. As the user activity is non-predictable over seconds or minutes, the model continuously trains to adapt to changing event patterns. During the training phase, raw event log

streams are fed into an online feature extractor which aggregates them into desired feature vectors to feed the DNN model. In the testing phase, the authors evaluated their approach using the CERT Insider Threat Dataset v6.2 and Tensorflow tool for implementation. As a result, the model achieved a better performance than existing approaches (Principal Component Analysis, SVM, Forest) with an average DR of 95.53%.

1.6 Challenges, Discussions and Conclusion

In this section, we present IDS challenges and potential solutions for each domain. We also discuss about the surveyed papers and provide some strategies to improve IDS in terms of accuracy, performance and robustness.

1.6.1 IDS challenges and potential solutions

Improving IDSs against evolving attacks must take into account the target domain behaviour, its dynamic and constraints. Domain-dependent IDS approaches are increasingly adopted as they are designed to support domain constraints and defend against domain-specific attacks. Such approaches are often more accurate than those independent from the domain (generic IDS). IDS challenges can also be generic and domain-specific. The generic challenges are common to all domain specific IDSs. In the literature, We have identified the 10 generic challenges below.

1) IDSs are not yet scaled to meet the requirements posed by high-speed networks (terabit, petabit) [122, 162], i.e. high-availability, low-latency, high-resiliency, high-reliability, and high-scalability.

2) Factors, like noise in the ingoing traffic, significantly affect traffic profiles, and the large amount of network traffic makes IDSs unable to build a normal traffic profile of the network for intrusion detection [252].

3) IDSs in default setting generate so many false positives [165] and fine-tuning the IDS can be an arduous undertaking that requires both time and expertise.

4) IDSs have not yet reached the level where they effectively give historical analysis of the intrusions detected over a period of time [281]. Thus, it is done manually by

cyber-analysts.

5) IDSs sometimes drop traffic when they become overloaded. As traffic levels rise, the associated processing load required to keep up becomes prohibitive and the IDS either falls behind or fails [71].

6) Some IDSs are ressource-intensives, i.e. they consume a lot of network bandwidth and requires high performance hardwares.

7) Many IDSs don't have the ability to decrypt the encrypted packets that they catch. When an IDS catches an encrypted packet, it typically discards it and attacks could be missed [21].

8) The evaluation of IDSs against the most recent attacks are not yet possible since DARPA, KDD cup 99, and NSL-KDD are deprecated. Whenever ethical hacking is done for penetration testing, it requires a lot of expertise, time and a deep knowledge of the target network.

9) Despite many unified event formats proposed, each IDS vendor still provide different event formats. This causes interoperability problems during processing.

10) As the frequency of signature updates varies from vendor to vendor, this may cause a problem for knowledge-based IDSs that require continuous updates for zero-day attack detection.

On the other hand, we have identified numerous IDS challenges for each of the surveyed domain (see Table 1.7). Domain-specific IDSs are subjected to many domain-constraints such as low-energy/low-memory consumption, minimal failures, dynamic topology, shorter downtimes, and human safety. The common baseline of the domain-specific IDSs is to ensure integrity, availability, confidentiality and authenticity of the information. To satisfy these requirements, IDSs must support cryptographic and trust management mechanisms [297] for data gathering, data processing and data sharing [182]. For example, IDSs could support the Datagram Transport Layer Security (DTLS) protocol designed for IPv6 over Low power Wireless Personal Area Networks (6LoWPANs) [245] and the key agreement protocol based on the elliptic curve cryptosystem to ensure integrity and mutual authentication among sensor nodes, gateway nodes and users [337].

Table 1.7 – Domain-specific Challenges

Parent Domains	Domains	Challenges
Industrial systems	Smart Grids	(1) IDSs do not yet support situational awareness for finer-grained command and control [343] of both the network traffic, energy distribution, power consumption monitoring, and physical components (e.g. substations, meters, sensors, computers); (2) IDSs must be able to prevent from false data injection, information leakage to external source, a fault or breakdown of a unit as generation, transmission, distribution or substation operational failure [182]; (3) IDSs must support distributed threat intelligence to continuously monitor smart metering infrastructures, micro-grids, and virtual power plants [44, 89, 200];
Health systems	Smart Health	(1) This field being extremely sensitive, IDSs must preserve security and privacy of patient's data from data modification and unavailability that may result in the patient's death; [52,168] (2) As recent health systems support various networking standards (e.g. Wifi, Bluetooth, Zigbee), IDSs should support end-to-end security of the electronic health record (EHR) and context-awareness for EHR monitoring; and (3) IDSs must provide a traceability of all access to healthcare data [10, 304]
Building systems	Smart Homes	(1) Smart homes support a plethoria of vulnerable networking technologies (Bluetooth, Zigbee, Wifi, Insteon, ZWave, Ethernet, RS485) and then, IDSs must support trust verification/-validation, end-to-end security, key management mechanisms (e.g. PKI), and continuous authentication schemes [182, 290]; (2) IDSs must adapt to low-power/low-memory constraints and must provide an end-point security of the connected devices; (3) Smart homes suffer from the heterogeneity of protocols due to different manufacturers/networking standards/firmware updates, IDSs should be continuously updated to handle multiple heterogeneous streams
Transportation	Smart Vehicles	(1) IDSs must adapt to the network size, the geography relevancy, the high mobility and dynamic topology, the short connection duration and the frequent disconnections; (2) IDSs must support trust and information verification mechanisms to secure connectivity between On-Board Units (vehicles) and Road-Side-Units (RSUs); (3) IDSs should provide key distribution mechanisms and efficient packet forwarding algorithms [164, 209, 220]

In Smart Health, IDSs could integrate the signature-based access control mechanism proposed in [8] to prevent unauthorized users from accessing patient data in the cloud. The mechanism supports a SaaS (Software-As-A-Service) application where only authorized professionals can access patient data. In Smart Vehicles, availability is one of the key factor due to high vehicular mobility. During V2I and V2V communication, messages are forwarded to a destination using multiple intermediate vehicles as relay nodes and forwarding algorithms such as Greedy Perimeter Stateless Routing (GPSR) [173] and new path recovery mechanism [192]. More challenges and solutions have been addressed in Smart Grids [44, 89, 171, 182, 200, 221, 244, 294, 343], Smart Health [10, 52, 168, 304], Smart Homes [163, 182, 229, 290, 351] and Smart Vehicles [102, 132, 164, 209, 220, 259, 287].

1.6.2 Evaluation of the surveyed work

In Table 1.4, anomaly-based detection techniques are most frequent in recent work and achieved high intrusion detection accuracies in the intervall [84.12%, 99.81%]. Moreover, hybrid detection techniques achieved a good performance with a high intrusion detection accuracy in the interval [84.84%, 99.91%]. Consequently, it is necessary to combine multiple detection approaches to increase the intrusion detection performance while having a look to the cost in time and space. Other techniques like knowledge-based and multi-agent based detection were coupled with/without anomaly detection to enhance accuracy.

Studied papers used NSL-KDD and KDD cup 1999 to evaluate their approaches. Precisely, 12.7% used NSL-KDD and 15.9% used KDD cup 1999. Other datasets were used by 22.15% of studied papers. Some papers evaluated their approach using a real-world environment (e.g. Industry tests), a self-built test bed and simulated environment (e.g. traffic generator, MATLAB): 14.3% of studied papers tested their solutions using a self-built test bed, 6.35% deployed the models in industry with real-world data, and 28.6% performed various simulations to show the effectiveness of their approaches. These statistics show that many researchers still used datasets and simulations rather than real deployment tests. As mentioned in Section 1.6.1, these datasets are deprecated as they do not take into account new attacks [350].

Researchers use deprecated datasets due to many reasons such as the building of a test environment is often complex and time consuming, professional skills or background experience is often required to set up the test environment, and cybersecurity companies do not allow researchers to perform internal real tests due to data privacy and laws.

Several papers on ESP approaches have been surveyed. An effective validation of these approaches requires a lot of resources. Hardware requirements and background experience in the ESP field limited considerably academic researches. However, the ESP necessity in big data security [350] is undeniable. In the studied papers, some authors combined different ESP approaches to enhance the performance of their solutions and efficiently reduce security event streams.

Chapitre 2

Extension des ASTDs

Résumé

Les diagrammes d'états-transitions algébriques (ASTD) sont des extensions d'automates et des diagrammes d'états communs qui peuvent être combinés avec des opérateurs d'algèbre de processus comme la séquence, le choix, la garde et la synchronisation quantifiée. Ils étaient auparavant introduits pour la représentation graphique, la spécification et la preuve des systèmes d'information. Dans le but d'utiliser les ASTD pour spécifier la détection des cyberattaques, nous avons identifié un certain nombre de fonctionnalités manquantes dans les ASTD. Cet article étend la notation ASTD avec des variables d'état (attributs), des actions sur les transitions et un nouvel opérateur appelé flux qui correspond aux états ET dans les diagrammes d'états et est un compromis entre l'entrelacement et la synchronisation dans les algèbres de processus. Nous définissons une sémantique opérationnelle structurée formelle de ces extensions et illustrons son implémentation dans un interpréteur basé sur OCaml appelé *iASTD* et le vérificateur de modèle ProB. Les ASTD étendus sont illustrés par une étude de cas sur la

détection des attaques cybernétiques.

Commentaires

La contribution ici réside dans l'extension du langage ASTD et la définition formelle de sa sémantique. Cette notation permet de prendre en compte la déclaration d'attributs (i.e., variables d'état), la déclaration des actions contenant du code executable pour modifier les attributs au cours de l'exécution d'une transition et dans l'état de l'automate (entrée, séjour, sortie), et un nouvel opérateur ASTD appelé *Flow* permettant d'exécuter des évènements partagés sur plusieurs modèles ASTDs lorsque cela est possible. Les actions peuvent être exécutées également au niveau d'un ASTD lui-même, afin de facilement factoriser le code à exécuter pour chaque transition de l'ASTD. La notation proposée a été implémentée dans deux interpréteurs, un en Prolog avec ProB, et l'autre en Ocaml. ProB s'est avéré un ajout utile, car il donne accès à plusieurs fonctionnalités de contrôle de modèle déjà implémentées, telles que le contrôle du raffinement, le contrôle de détermination et la vérification de formules temporelles [234].

Les contributions décrites dans ce chapitre ont fait l'objet d'un article publié dans le cadre de la $23^{ième}$ édition de la conférence internationale *ICECCS (International Conference on Engineering of Complex Computer Systems)*, de rang A, qui a eu lieu à Melbourne, Australie, le 14 décembre 2018.

Les contributions et l'article sus-cité ont été élaborées par mes soins en tenant compte des remarques et commentaires issus de mon équipe d'encadrement.

Extended Algebraic State-Transition Diagrams

Lionel N. Tidjon
Université de Sherbrooke, GRIF, Québec, Canada
Télécom SudParis, SAMOVAR, Institut Polytechnique Paris, France
lionel.nganyewou.tidjon@usherbrooke.ca

Marc Frappier
Université de Sherbrooke, GRIF, Québec, Canada
marc.frappier@usherbrooke.ca

Michael Leuschel
Institut für Informatik, Universität Düsseldorf, Düsseldorf, Germany
leuschel@hhu.de

Amel Mammar
Télécom SudParis, SAMOVAR, Institut Polytechnique Paris, France
amel.mammar@telecom-sudparis.eu

Keywords: Formal Methods, Algebraic State-Transition Diagrams, Information Systems, Cyber Attack Detection

Abstract

Algebraic State-Transition Diagrams (ASTDs) are extensions of common automata and statecharts that can be combined with process algebra operators like sequence, choice, guard and quantified synchronization. They were previously introduced for the graphical representation, specification and proof of information systems. In an attempt to use ASTDs to specify cyber attack detection, we have identified a number of missing features in ASTDs. This paper extends the ASTD notation with state variables (attributes), actions on transitions, and a new operator called flow which corresponds to AND states in statecharts and is a compromise between interleaving and synchronization in process algebras. We provide a formal structured operational semantics of these extensions and illustrate its implementation in an OCaml-based interpreter called iASTD and the model checker ProB. Extended ASTDs are illustrated in a case study in cyber attack detection.

2.1 Introduction

Model-based languages like Abstract State Machines (ASM) [49], B [3] and Z [296] provide rich environments for specifying data and behavioural aspects of systems. However, the modelling of control flow in these languages, e.g., sequence, choice or parallel composition, is not easy to understand and validate with users. Process algebras like Communicating Sequential Processes (CSP) [142] and the Calculus of Communicating Systems (CCS) [125] describe interactions, communications, and synchronizations between processes. They support rich operators like sequence, choice, parallel composition, interleaving and iteration. But, they do not support a concise and elegant way to describe complex data aspects [243].

Combinations of CSP and Z (e.g., Circus [325]), CSP and B (e.g. CSP‖B [303], CSP2B [51]) integrate process algebras with model-based notations to provide a richer specification environment that provides a more explicit representation of the control flow and support a rich notation for data modelling. On the other hand, graphical notations like statecharts [130, 141] and their variants offer an even more explicit representation of control and have shown their usefulness in various domains. Algebraic-State Transition Diagrams (ASTDs) were proposed in [97] to combine the graphical strengths of statecharts and the abstraction power of process algebra operators. An elementary ASTD is an automaton. Elementary ASTDs can be easily integrated with operators like sequence, Kleene closure, choice, guard, parallel composition with synchronization, quantified choice and quantified parallel composition. Like in statecharts, automaton states can themselves be complex ASTDs. ASTDs can be translated into B for formal analysis and proof [216]. They can be associated with B machines to model data and they can be refined in the Event-B style by adding new events [93, 99]. A B machine contains one operation for each event and it encapsulates data associated to the ASTD, similar to the CSP2B approach. When an event is executed by the ASTD, the corresponding operation in the data B machine is also executed.

In this paper, we propose an extension of the ASTD notation to support the declaration of attributes (i.e., state variables) and actions that can modify these attributes when a transition is executed. In contrast to [51, 93, 303], these extensions

enable to support data handling directly within the ASTD specification, closer to the statecharts style. Thus, the proposed extension provides the same level of data modelling as in tools like Stateflow [145], but with the additional control flow abstraction capability of process algebra operators. Attributes can be locally declared within each ASTD. Actions can be executed on automaton transitions, but also at the level of an ASTD itself, in order to easily factor out code that needs to be executed for every transitions of an ASTD. This paper proposes also a new ASTD operator called *flow* which corresponds to an AND state in statecharts and is a compromise between interleaving and synchronization in process algebras. It combines multiple ASTDs and executes an event on each of them that can execute it. Thus, it is a form of *weak* synchronization. We have identified the need for these extensions while exploring the use of ASTDs to model cyber security attacks.

This paper provides a formal operational semantics of these extensions. It also presents an interpreter implemented in OCaml and a model checker based on ProB [187] that can be used to execute and validate ASTD specifications.

The rest of this paper is structured as follows. Section 2.2 introduces the extended ASTD notation and describes its operational semantics. Section 2.3 presents a case study in cyber attack detection and illustrates the proposed extensions. In Section 2.4, we present the ASTD interpreter and the model checker based on ProB. Section 2.5 concludes by some discussions and perspectives.

2.2 Extended ASTD Syntax and Semantics

This section aims to formally specify the extensions (i.e., attributes, actions and the new operator flow). Actions and attributes are required to more precisely and concisely describe cyber attacks [199], while the new operator allows executing multiple attack events from various sources (e.g., network, host) and combining different attack models, being parts of a whole attack. This is particularly useful to detect multi-step attacks and advanced persistent threats that originate from multiple entry points and attack vectors [1]. Introducing attributes and actions require adding a concept of global state to the ASTD semantic rules, while the new operator "simply"

1. https://www.fireeye.com/current-threats/anatomy-of-a-cyber-attack.html

requires its own rules to be added, albeit with a twist (Sect. 2.2.4).

The structure (syntax) of ASTDs is defined using a type hierarchy. Each ASTD operator is represented by a type. To identify characteristics shared by all ASTD types, we define an abstract ASTD type $\mathsf{ASTD} \triangleq \langle n, P, V, A_{astd} \rangle$ where $n \in \mathsf{Name}$ is the name of the ASTD, P is an optional list of parameters, V is a set of attributes, $A_{astd} \in \mathcal{A}$ is an action; its default value is **skip**, which does nothing. Parameters P are used to receive values passed by a calling ASTD; they can be read-only or read-write. Attributes V are state variables that can be modified by actions and tested in guards within the scope of the ASTD. Actions can also modify attributes received as parameters of the ASTD. Each ASTD type inherits from the supertype ASTD.

We also distinguish between the syntax of an ASTD and its state. We denote by States the set of states of ASTDs. Each ASTD type gives rise to a subtype of States. In this paper, we content ourselves with the definition of the following subtypes of ASTDs: elem, Sequence, Automaton, Synchronization and Flow. A complete definition of extended ASTDs is available at [310]. Final states of an ASTD are determined by a function *final* of type $\mathsf{ASTD} \times \mathsf{States} \rightarrow \mathsf{Boolean}$. Function *init* of type $\mathsf{ASTD} \rightarrow \mathsf{States}$ returns the initial state of an ASTD.

The semantics of ASTDs consists of a labeled transition system (LTS). A LTS is a subset of $\mathsf{States} \times \mathsf{Event} \times \mathsf{States}$ and a transition is denoted by $s \xrightarrow{\sigma}_a s'$. It means that ASTD a can execute event σ from state s and move to state s'. The semantics of an ASTD depend on the variables declared in its enclosing ASTDs; we use environments to represent the values of these variables. An environment is a partial function of type $\mathsf{Env} \triangleq \mathsf{Var} \nrightarrow \mathsf{Term}$ which assigns values to variables. We need to introduce an auxiliary transition relation that handles environments:

$$s \xrightarrow{\sigma, E_e, E'_e}_a s'$$

where E_e, E'_e denote the before and after values of variables in the ASTDs enclosing ASTD a. The first rule provided below relates the state-transition relation with the auxiliary one. It states that a transition is proved starting with empty environments.

$$\mathsf{env}\ \frac{s \xrightarrow{\sigma,\{\},\{\}}_a s'}{s \xrightarrow{\sigma}_a s'}$$

ASTDs are *non-deterministic*. If several transitions on σ are possible from a given state s, then one of them is non-deterministically chosen. The operational semantics is inductively defined in the sequel for some ASTD subtypes.

2.2.1 Automaton

2.2.1.1 Syntax

An automaton ASTD is a structure $\mathsf{Automaton} \triangleq \langle \mathsf{aut}, \Sigma, S, \zeta, \nu, \delta, SF, DF, n_0 \rangle$ with the following constraints. $\Sigma \subseteq \mathsf{Event}$ is the alphabet. $S \subseteq \mathsf{Name}$ is the set of state names. $\zeta \in S \to \langle A_{in}, A_{out}, A_{stay} \rangle$ maps each state name to its actions: A_{in} is executed when a transition enters the state; A_{out} is executed when a transition leaves the state; A_{stay} is executed when a transition loops on the state or is executed within the state. $\nu \in S \to \mathsf{ASTD}$ maps each state name to its sub-ASTD, which can be elementary (noted elem) or complex. An automaton transition between states $n_1, n_2 \in S$ labeled with $\sigma[g]/A_{tr}$ is represented as a tuple in the transition relation δ as follows:

$$((\mathsf{loc}, n_1, n_2), \sigma, g, A_{tr}, \mathit{final}?) \in \delta$$

Symbol *final*? is a Boolean: when *final*? = true, the source of the transition is decorated with a bullet; it indicates that the transition can be fired only if n_1 is final. The *final*? field is only useful when n_1 is a complex state. We also write $\delta((\mathsf{loc}, n_1, n_2), \sigma, g, A_{tr}, \mathit{final}?)$ to state that a tuple is an element of δ.

There are other types of automaton transitions (e.g., to, or from, a state of a nested automaton); they are omitted here; see [310] for further information. $SF \subseteq S$ is the set of shallow final states, while $DF \subseteq S$ denotes the set of deep final states, with $DF \cap SF = \varnothing$. $n_0 \in S$ is the name of the initial state. The definition of $final$, provided in the sequel, will describe the distinction between the two types of final states.

The state of an automaton cannot be simply represented by a state name. It is a more complex structure of type $\langle \mathsf{aut}_\circ, n, E, h, s \rangle$. $\mathsf{aut}_\circ$ is the constructor of the automaton state. $n \in S$ denotes the current state of the automaton. E contains the values of the automaton attributes. $h \in S \nrightarrow \mathsf{States}$ is the history function that

implements the notion of *history state* used in statecharts; it records the last visited sub-state of a state. $s \in \mathsf{States}$ is state of the sub-ASTD of n, when n is a complex state; $s = \mathsf{elem}$ when n is elementary.

Given an automaton $a \in \mathsf{Automaton}$, we denote by $a.Field$ with $Field \in \{\Sigma, S, \zeta, \nu, \delta, SF, DF, n_0\}$ the corresponding component of the tuple, i.e., $a.n_0$ denotes the initial state of a.

Functions *init* and *final* are now defined as follows. Let a be an automaton ASTD.

$$\begin{aligned} init(a) &\triangleq (\mathsf{aut}_\circ, a.n_0, a.E_{init}, h_{init}, init(a.\nu(n_0))) \\ h_{init} &\triangleq \{n \mapsto init(a.\nu(n)) \mid n \in a.S\} \\ final(a, (\mathsf{aut}_\circ, n, E, h, s)) &\triangleq n \in a.SF \vee (n \in a.DF \wedge final(a.\nu(n), s)) \end{aligned}$$

Symbol E_{init} denotes the initial values of attributes, as specified in their declaration. $init(a.\nu(n_0))$ returns the initial state of the sub-ASTD $\nu(n_0)$ of n_0. For example, in Fig. 2.1, $init(A.\nu(1)) = \mathsf{elem}$ as state 1 is elementary and $init(\mathsf{A}) = (\mathsf{aut}_\circ, 1, \{(x, 0)\}, h_{init}, \mathsf{elem})$, where $h_{init} = \{1 \mapsto \mathsf{elem}\}$. Symbol h_{init} is the initial value of the history function; it maps each state name to the initial state of its internal structure: elementary states are mapped to the constant elem (i.e., $init(\mathsf{elem}) = \mathsf{elem}$); complex automaton states are mapped to the initial state of their sub-ASTD, recursively. A deep final state is final only when its sub-ASTD is also final, whereas a shallow final state is final irrespective of the state of its sub-ASTD.

2.2.1.2 Semantics

There are six rules of inference to define the semantics of an automaton, in order to deal with the different types of transitions and states. The two most frequently used rules are illustrated here. The other rules are defined in [310].

The first rule, aut_1, describes a transition between local states.

$$\mathsf{aut}_1 \ \frac{a.\delta((\mathsf{loc}, n_1, n_2), \sigma', g, A_{tr}, final?) \quad \Psi \quad \Omega_{loc}}{(\mathsf{aut}_\circ, n_1, E, h, s_1) \xrightarrow{\sigma, E_e, E'_e}_a (\mathsf{aut}_\circ, n_2, E', h', init(a.\nu(n_2)))}$$

The conclusion of this rule states that a transition on event σ can occur from n_1 to n_2 with before and after automaton attributes values E, E'. The state of the sub-ASTD of n_2 is its initial state (i.e., $init(a.\nu(n_2))$). The premiss provides that such

a transition is possible if there is a matching transition in δ, which is represented by $\delta((\mathsf{loc}, n_1, n_2), \sigma', g, A_{tr}, \mathit{final}?)$. σ' is the event labelling the transition, and it may contain variables. The value of these variables is given by the environment E_e and local attributes values E, which can be applied as a substitution to a formula using operator $(\![\ ]\!)$. This match on the transition is provided by premiss Ψ defined as follows.

$$\Psi \triangleq \Big((\mathit{final}? \Rightarrow final(a, (\mathsf{aut}_\circ, n_1, E, s))) \ \wedge \ g \ \wedge \ \sigma' = \sigma\Big)(\![E_g]\!)$$

Ψ can be understood as follows. If the transition is final (i.e., $\mathit{final}? = \mathsf{true}$), then the current state must be final. The transition guard g holds. The event received, noted σ, is equal to the event σ' which labels the automaton transition, after applying the environment E_g as a substitution. Environment E_g is defined in premiss Ω_{loc}.

$$\Omega_{loc} \triangleq \left\{ \begin{array}{l} \textbf{if } n_1 = n_2 \textbf{ then} \\ \quad A = A_{tr} \ ; \ a.\zeta(n_1).A_{stay} \ ; \ a.A_{astd} \\ \textbf{else} \\ \quad A = a.\zeta(n_1).A_{out} \ ; \ A_{tr} \ ; \ a.\zeta(n_2).A_{in} \ ; \ a.A_{astd} \\ \textbf{end} \\ E_g = E_e \lhd\!\!\!\lhd E \\ A(E_g, E'_g) \\ E'_e = E_e \lhd\!\!\!\lhd (a.V \lhd\!\!\!| E'_g) \\ E' = a.V \lhd\!\!\!| E'_g \\ h' = h \lhd\!\!\!\lhd \{n_1 \mapsto s_1\} \end{array} \right\}$$

Premiss Ω_{loc} uses the relational domain restriction operator $U \lhd\!\!| r = \{x \mapsto x' \mid x \in U \wedge x \mapsto x' \in r\}$, where r is a relation and U a set, and the domain subtraction $U \lhd\!\!\!| r = \{x \mapsto x' \mid x' \notin U \wedge x \mapsto x' \in r\}$, and the override $r_1 \lhd\!\!\!\lhd r_2 = (dom(r_2) \lhd\!\!\!| r_1) \cup r_2$. The execution of an action A on attributes E with possible after value E' is noted $A(E, E')$. The sequential execution of actions A_1 and A_2 is noted $A_1 \ ; \ A_2$. Premiss Ω_{loc} can be understood as follows. The actions executed are the transition action A_{tr}, followed by the stay action $\zeta(n_1).A_{stay}$ of state n_1 and finally the ASTD action $a.A_{astd}$, which is declared in the heading of the automaton. The ASTD action is useful to factor out state modifications that must be done on every transition of the

ASTD. State actions (entry, exit and stay) are useful to factor out modifications that must done on all transitions upon entry, exit or loop, of a given state. When the transition is not a loop (i.e., $n_1 \neq n_2$), the actions executed are the exit code of n_1, the transition action, the entry code of n_2 and finally the ASTD action. Symbol E_g, defined as $E_e \lhd\!\!\lhd E$, denotes the global list of variables that can be modified by the actions. It includes the variables declared in the enclosing ASTDs (E_e) and the variables declared locally (E). When a local variable bears the same name as a variable declared in the enclosing ASTDs, it overrides it, similarly to shadowing in programming languages like C, which is represented using the override operator ($\lhd\!\!\lhd$). Their after values E'_g are used to set E' (the local attributes) using the restriction on the attributes V declared in the ASTD and the values E'_e (the attributes declared in enclosing ASTDs), to model variable shadowing.

Rule aut_6, handles transitions within the sub-ASTD $a.\nu(n)$ of state n.

$$\mathsf{aut}_6 \frac{s \xrightarrow{\sigma, E_g, E''_g}_{a.\nu(n)} s' \qquad \Theta}{(\mathsf{aut}_\circ, n, E, h, s) \xrightarrow{\sigma, E_e, E'_e}_{a} (\mathsf{aut}_\circ, n, E', h, s')}$$

$$\Theta \triangleq \begin{pmatrix} E_g = E_e \lhd\!\!\lhd E \quad a.A_{astd}(E''_g, E'_g) \\ E'_e = E_e \lhd\!\!\lhd (V \lhd E'_g) \quad E' = V \lhd E'_g \\ \end{pmatrix}$$

The transition starts from a sub-state s and moves to the sub-state s' of state n. Actions are executed bottom-up. E''_g denotes the values computed by the sub-ASTD. Premiss Θ defines the computation of E'_g from E''_g by executing the ASTD action A_{astd}. E'_e and E' are extracted by partitioning E'_g using V. Premiss Θ is reused in all subsequent rules where a sub-ASTD transition is involved.

2.2.1.3 Example

Fig. 2.1 provides an automaton ASTD that we can use to illustrate transitions and transition execution proofs. Automaton A declares an attribute set V={(x,int,0)} which contains attribute x, of type int with initial value 0. Automaton A also declares an action x:=!x+2. Actions are expressed in OCaml; expression !x denotes the before

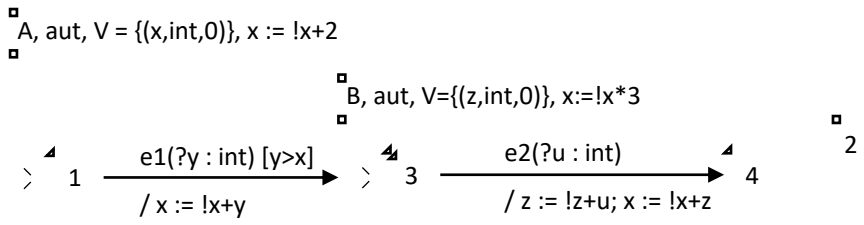


Figure 2.1 – An automaton ASTD with a complex state 2

value of x. State 1 is an elementary state depicted by ⟜○. State 2 of ASTD A is a complex state, the automaton ASTD B. The transition from 1 to 2 is labeled with event e1(?y : int), which declares a local variable y whose scope is only the transition. It also contains guard [y > x] and an action x:=!x+y. ASTD B, of state 2, declares a local variable z and an ASTD action x := !x∗3. Its initial state is also final and it is computed by the function $init(\nu(2)) = (\mathsf{aut}_\circ, 3, \{(z,0)\}, \mathsf{elem})$.

ASTD A can execute the following two transitions.

$$\begin{aligned} & (\mathsf{aut}_\circ, 1, \{(x,0)\}, \mathsf{elem}) \\ \xrightarrow{e1(1)}_A \quad & (\mathsf{aut}_\circ, 2, \{(x,3)\}, (\mathsf{aut}_\circ, 3, \{(z,0)\}, \mathsf{elem})) \\ \xrightarrow{e2(1)}_A \quad & (\mathsf{aut}_\circ, 2, \{(x,14)\}, (\mathsf{aut}_\circ, 4, \{(z,1)\}, \mathsf{elem})) \end{aligned}$$

The proof of the first transition is the following, stripping the keywords $\mathsf{aut}_\circ$ and elem for the sake of concision.

$$\mathsf{env}\ \frac{\mathsf{aut}_1\ \dfrac{\delta((\mathsf{loc},1,2), e1(y), y>x, x:=!x+y, \mathsf{false}) \quad (y>x \wedge e1(y) = e1(1))([\![x:=0]\!])}{(1,\{(x,0)\}) \xrightarrow{e1(1),\{\},\{\}}_A (2,\{(x,3)\},(3,\{(z,0)\}))}}{(1,\{(x,0)\}) \xrightarrow{e1(1)}_A (2,\{(x,3)\},(3,\{(z,0)\}))}$$

Rule env adds the empty environments. Rule aut_1 succeeds, because y is valued to 1 by the equality $e1(y) = e1(1)$ and the guard $y > x$ holds after substituting x with 0. The symbols of premiss Ω_{loc} in step aut_1 are valued as follows.

$$E = \{(x,0)\} \quad E_e = \{\}$$
$$E_g = \{\} \lhd\!\!+ \{(x,0)\} = \{(x,0)\}$$
$$E'_g = \{(x,3)\},$$
$$\text{since}\{\mathsf{x{:=}!x{+}1;\ x\ :=\ !x{+}2}\}(\{(x,0)\},\{(x,3)\})$$
$$E'_e = \{\} \lhd\!\!+ (\{x\} \lhd \{(x,3)\}) = \{\}$$
$$E' = \{x\} \lhd \{(x,3)\} = \{(x,3)\}$$

The proof of the second transition is the following.

$$\dfrac{\dfrac{\dfrac{\delta((\mathsf{loc},3,4),e2(u),\mathsf{true},z:=!z+u;x:=!x+z,\mathsf{false})}{(e2(u)=e2(1))([x:=3,z:=0])}\ \mathsf{aut}_1}{(3,\{(z,0)\}) \xrightarrow{e2(1),\{(x,3)\},\{(x,12)\}}_B (4,\{(z,1)\})}\ \mathsf{aut}_6}{(2,\{(x,3)\},(3,\{(z,0)\})) \xrightarrow{e2(1),\{\},\{\}}_A (2,\{(x,14)\},(4,\{(z,1)\}))}\ \mathsf{env}$$
$$(2,\{(x,3)\},(3,\{(z,0)\})) \xrightarrow{e2(1)}_A (2,\{(x,14)\},(4,\{(z,1)\}))$$

2.2.2 Sequence

The sequence ASTD allows for the sequential composition of two ASTDs. When the first item reaches a final state, the second one can start its execution [310]. This enables decomposing problems into a set of tasks that have to be executed in sequence.

2.2.2.1 Syntax

A sequence ASTD is a structure $\langle \fatsemi, fst, snd\rangle$ where fst, snd are ASTDs denoting respectively the first and second sub-ASTDs of the sequence. A sequence state is of type $\langle \fatsemi_\circ, E, [\mathsf{fst} \mid \mathsf{snd}], s\rangle$, where $\fatsemi_\circ$ is a constructor of the sequence state, E the values of attributes declared in the sequence, $[\mathsf{fst} \mid \mathsf{snd}]$ is a choice between two markers that respectively indicate whether the sequence is in the first sub-ASTD or the second sub-ASTD and $s \in \mathsf{States}$. Functions *init* and *final* are defined as follows. Let a be a

sequence ASTD.

$$
\begin{aligned}
\mathit{init}(a) &\triangleq (\mathrel{⇨}_\circ, a.E_{init}, \mathsf{fst}, \mathit{init}(a.fst)) \\
\mathit{final}(a, (\mathrel{⇨}_\circ, E, \mathsf{fst}, s)) &\triangleq \mathit{final}(a.fst, s) \quad \wedge \\
&\qquad \mathit{final}(a.snd, \mathit{init}(a.snd)) \\
\mathit{final}(a, (\mathrel{⇨}_\circ, E, \mathsf{snd}, s)) &\triangleq \mathit{final}(a.snd, s)
\end{aligned}
$$

The initial state of a sequence is the initial state of its first sub-ASTD. A sequence state is final when either i) it is executing its first sub-ASTD and this one is in a final state, and the initial state of the second sub-ASTD is also a final state; ii) it is executing the second sub-ASTD which is in a final state.

2.2.2.2 Semantics

Three rules are necessary to define the execution of the sequence. Rule $⇨_1$ deals with transitions on the sub-ASTD *fst* only. Rule $⇨_2$ deals with transitions from *fst* to *snd*, when *fst* is in a final state. Rule $⇨_3$ deals with transitions on the sub-ASTD *snd*.

$$
⇨_1 \frac{s \xrightarrow{\sigma, E_g, E''_g}_{a.fst} s' \qquad \Theta}{(\mathrel{⇨}_\circ, E, \mathsf{fst}, s) \xrightarrow{\sigma, E_e, E'_e}_{a} (\mathrel{⇨}_\circ, E', \mathsf{fst}, s')}
$$

$$
⇨_2 \frac{\mathit{final}(a.fst, s)(\llbracket E_g \rrbracket) \quad \mathit{init}(a.snd) \xrightarrow{\sigma, E_g, E''_g}_{a.snd} s' \quad \Theta}{(\mathrel{⇨}_\circ, E, \mathsf{fst}, s) \xrightarrow{\sigma, E_e, E'_e}_{a} (\mathrel{⇨}_\circ, E', \mathsf{snd}, s')}
$$

$$
⇨_3 \frac{s \xrightarrow{\sigma, E_g, E''_g}_{a.snd} s' \qquad \Theta}{(\mathrel{⇨}_\circ, E, \mathsf{snd}, s) \xrightarrow{\sigma, E_e, E'_e}_{a} (\mathrel{⇨}_\circ, E', \mathsf{snd}, s')}
$$

2.2.3 Parameterized Synchronization

Synchronization ASTDs allow managing concurrent resources between two ASTD components using the synchronization operator [310]. Events and variables between the two components are recorded in a synchronization set Δ for handling synchronization.

2.2.3.1 Syntax

A parameterized synchronization ASTD is a structure $\langle |[]|, \Delta, l, r \rangle$ where Δ is the synchronization set of event labels, $l, r \in \mathsf{ASTD}$ are the synchronized ASTDs. When the label of the event belongs to Δ, the two sub-ASTDs must both execute it; otherwise either the left or the right sub-ASTD can execute it; if both sub-ASTDs can execute it, the choice between them is nondeterministic. When $\Delta = \varnothing$, the synchronization is called an interleaving, noted $|||$.

A parameterized synchronization state is of type $\langle |[]|_\circ, E, s_l, s_r \rangle$, where s_l, s_r are the states of the left and right sub-ASTDs. Initial and final states are defined as follows. Let a be a parameterized synchronized ASTD.

$$\begin{aligned} \mathit{init}(a) &\triangleq (|[]|_\circ, a.E_{init}, \mathit{init}(a.l), \mathit{init}(a.r)) \\ \mathit{final}(a, (|[]|_\circ, E, s_l, s_r)) &\triangleq \mathit{final}(a.l, s_l) \wedge \mathit{final}(a.r, s_r) \end{aligned}$$

2.2.3.2 Semantics

There are three inference rules. Rules $|[]|_1$ and $|[]|_2$ respectively describe execution of events, with no synchronization required, either on the left or the right sub-ASTDs. Rule $|[]|_1$ below caters for execution on the left sub-ASTD. The function $\alpha(e)$ returns the label of event e.

$$|[]|_1 \ \frac{\alpha(\sigma) \notin \Delta \qquad s_l \xrightarrow{\sigma, E_g, E''_g}_{a.l} s'_l \qquad \Theta}{(|[]|_\circ, E, s_l, s_r) \xrightarrow{\sigma, E_e, E'_e}_{a} (|[]|_\circ, E', s'_l, s_r)}$$

Rule $|[]|_2$ is symmetric to $|[]|_1$ and indicates behaviour when the right side execute the action.

$$|[]|_2 \ \frac{\alpha(\sigma) \notin \Delta \qquad s_r \xrightarrow{\sigma, E_g, E''_g}_{a.r} s'_r \qquad \Theta}{(|[]|_\circ, E, s_l, s_r) \xrightarrow{\sigma, E_e, E'_e}_{a} (|[]|_\circ, E', s_l, s'_r)}$$

The most interesting case is when the left and right sub-ASTDs must synchronize on an event (i.e., when $\alpha(\sigma) \in \Delta$). Consider the transitions on the left and right sub-ASTDs when each of them is executed independently of the other.

$$\Omega_{ilr} \triangleq \left(s_l \xrightarrow{\sigma, E_g, E'_{gl}}_{a.l} s'_l \qquad s_r \xrightarrow{\sigma, E_g, E'_{gr}}_{a.r} s'_r \right)$$

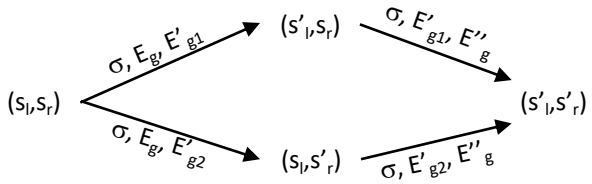


Figure 2.2 – Commutativity of actions execution in a parameterized synchronization on σ

Since shared variables (E_g) can be modified by both sub-ASTDs, their modifications could be inconsistent, which should forbid the synchronization transition. This requires to check that $E'_{gl} = E'_{gr}$. However, when one variable is modified by both sub-ASTDs, the natural intent is typically that the compound result of both sides is desired, that is, to assume that one side executes on the values returned by the other. For instance, assume that both sub-ASTDs increment shared variable x by 1 and assume that the before value of x is 0. The above semantics gives $x = 1$, whereas the compound result is $x = 2$. If one sub-ASTD increments by 1 and the other by 2, the above semantics forbids execution because the result is inconsistent, whereas the compound execution returns 3. Executing the left sub-ASTD before the right sub-ASTD is specified as follows.

$$\Omega_{lr} \triangleq \left(s_l \xrightarrow{\sigma,E_g,E'_{g1}}_{a.l} s'_l \qquad s_r \xrightarrow{\sigma,E'_{g1},E''_g}_{a.r} s'_r \right)$$

Executing the right sub-ASTD before the left sub-ASTD is specified as follows.

$$\Omega_{rl} \triangleq \left(s_r \xrightarrow{\sigma,E_g,E'_{g2}}_{a.r} s'_r \qquad s_l \xrightarrow{\sigma,E'_{g2},E''_g}_{a.l} s'_l \right)$$

Since synchronization is commutative, i.e., both execution orders should return the same states. Fig. 2.2 illustrates this property.

Checking this commutativity at each transition is expensive. To improve performance, it could be statically checked using proof obligations, or by analysis of the variables read and written by each sub-ASTD, to ensure that the left and right sub-ASTDs are independent (i.e., they do not modify the same variables and one sub-ASTD does not modify the variables read by the other). When the synchronization operands are nondeterministic, this is still expensive to execute, because a commutative combination must be found, which means enumerating combinations

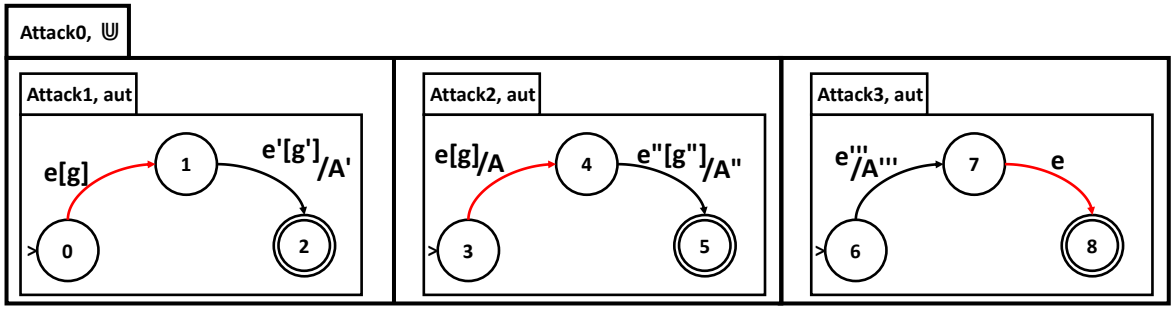


Figure 2.3 – Using the Flow operator to combine multiple attack models

of possibilities from both sides. Still, this is our preferred semantics for a synchronization, because it works well for deterministic specifications. Here is the rule for synchronization.

$$|[]|_3 \frac{\alpha(\sigma) \in \Delta \qquad \Omega_{lr} \qquad \Omega_{rl} \qquad \Theta}{(|[]|_\circ, E, s_l, s_r) \xrightarrow{\sigma, E_e, E'_e}_a (|[]|_\circ, E', s'_{l_1}, s'_{r_1})}$$

Note that our treatment of synchronization differs from those in CSP, CSP‖B, CSP2B or Circus. Since CSP is a pure process algebra, it does not support attributes. In CSP‖B and CSP2B, a single operation is executed to update a set of global state variables; the problem of having two different actions modifying the same variable does not exist. However, if the action to execute depends on the state of the CSP expression, extra state variables must be added to encode the CSP state in the B machine, which introduces some undesirable coupling between the CSP specification and the B specification, breaking the ideal separation between the control part represented in CSP and the data part represented in B. In Circus, synchronization occurs on channels only; state variables are not modified on a synchronization between two processes; actions are executed in interleave. The ASTD action is particularly useful for the synchronization ASTD, because it enables to make decisions and updates based on the result of the execution in both sub-ASTDs.

Also note that to process and combine action effects performed in sequence or in parallel, there are multiple alternatives available. The B [3] style parallel updates are quite restrictive, and disallow parallel assignments to the same variables. The ASM approach [49] is to collect parallel updates and perform them at the end of an (atomic) event. The translation from ASM to B in [188] proposes to use update functions which

are composed. While elegant, the actions in our ASTD implementation take effect immediately and are executed by a "black box" interpreter. Hence this was not a practical approach.

2.2.4 Flow

It is quite common that the same event e can be part of several cyber attack specifications, as illustrated in Fig. 2.3. An intrusion detection system needs to execute such an event on each attack specification that can execute it; this behaviour is not fulfilled by either interleaving or synchronization of attack specifications. This raises the need of a new operator $\Cup$, called flow, which will execute the event on each sub-ASTD whenever possible, like AND states in statecharts. In contrast to other ASTD operators, the rules for this operator involve negation, i.e., one has to determine whether an event is not possible for sub-ASTDs. In that respect it is related to FDR's priority annotation for CSP [103]. Such an annotation could probably be used to implement $\Cup$.

In Fig. 2.3, Attack0 combines multiple attacks using a flow: Attack1, Attack2 and Attack3. The flow operator checks all possible transitions from each attack component and executes them. When event e is received, transitions 0-1, 3-4 and 7-8 are executed if their ASTDs are in state 0, 3 and 7, respectively. Other transitions (i.e., 1-2, 4-5 and 6-7) execute in interleaving.

2.2.4.1 Syntax

A flow ASTD is a structure $\langle \Cup, l, r \rangle$. A flow state is of type $\langle \Cup_\circ, E, s_l, s_r \rangle$, where s_l, s_r are the states of the left and right sub-ASTDs. Initial and final states are defined as follows. Let a be a flow ASTD.

$$\begin{aligned} \mathit{init}(a) &\triangleq (\Cup_\circ, a.E_{\mathit{init}}, \mathit{init}(a.l), \mathit{init}(a.r)) \\ \mathit{final}(a, (\Cup_\circ, E, s_l, s_r)) &\triangleq \mathit{final}(a.l, s_l) \wedge \mathit{final}(a.r, s_r) \end{aligned}$$

2.2.4.2 Semantics

We use the following abbreviation to denote that an ASTD cannot execute a transition from a state s and global attributes E_g.

$$s \xrightarrow{\sigma, E_g}_a \bot \quad \triangleq \quad \neg \exists\, E'_g, s' \cdot s \xrightarrow{\sigma, E_g, E'_g}_a s'$$

Symbol $\bot$ is used to denote an undefined value. The negation of a transition predicate is computed using the usual negation as failure approach. For example (Fig. 2.3), the execution of transitions 0 to 1 and 3 to 4 fails on the reception of the event $\mathsf{e'''}$. Then, they are ignored while the transition 6 to 7 is executed. Here are the first two rules when only one of the two sub-ASTDs can execute the event.

$$\Uparrow_1 \frac{s_l \xrightarrow{\sigma, E_g, E''_g}_{a.l} s'_l \quad s_r \xrightarrow{\sigma, E''_g}_{a.r} \bot \quad \Omega_{lr} \Leftrightarrow \Omega_{rl} \quad \Theta}{(\Uparrow_\circ, E, s_l, s_r) \xrightarrow{\sigma, E_e, E'_e}_a (\Uparrow_\circ, E', s'_l, s_r)}$$

$$\Uparrow_2 \frac{s_r \xrightarrow{\sigma, E_g, E''_g}_{a.r} s'_r \quad s_l \xrightarrow{\sigma, E''_g}_{a.l} \bot \quad \Omega_{lr} \Leftrightarrow \Omega_{rl} \quad \Theta}{(\Uparrow_\circ, E, s_l, s_r) \xrightarrow{\sigma, E_e, E'_e}_a (\Uparrow_\circ, E', s_l, s'_r)}$$

The premiss $\Omega_{lr} \Leftrightarrow \Omega_{rl}$ ensures that one execution order succeeds iff the other also succeeds. It ensures the determinacy of the flow operator.

The third rule describes the case where both sub-ASTDs can execute the event; it is almost the same as $|[]|_3$, as it requires commutativity.

$$\Uparrow_3 \frac{\Omega_{lr} \qquad \Omega_{rl} \qquad \Theta}{(\Uparrow_\circ, E, s_l, s_r) \xrightarrow{\sigma, E_e, E'_e}_a (\Uparrow_\circ, E', s'_{l_1}, s'_{r_1})}$$

2.2.5 Choice

A choice ASTD allows a choice between two sub-ASTDs. Once a sub-ASTD has been chosen, the other sub-ASTD is ignored [310]. The choice is nondeterministic if each sub-ASTD can execute the requested event.

2.2.5.1 Syntax

The choice ASTD subtype has the following structure:

$$\mathsf{Choice} \triangleq \langle |, l, r \rangle$$

where $l, r \in \mathsf{ASTD}$ are respectively the first and second element of the choice. The type of a choice state is $\langle|_\circ, E, side, s\rangle$ where $side \in (\bot \mid \langle\mathsf{left}\rangle \mid \langle\mathsf{right}\rangle)$ denotes the sub-ASTD which has been chosen, $s \in (\mathsf{States} \mid \bot)$ denotes the state of the sub-ASTD which has been chosen and E the values of attributes declared in the choice ASTD. A choice state is final if i) it hasn't started yet and the initial state of each sub-ASTD is final, or ii) the chosen sub-ASTD is in a final state. Here are the formal definitions of the initial state and the final states. Let a be a choice ASTD.

$$
\begin{aligned}
init(a) &\triangleq (|_\circ, a.E_{init}, \bot, \bot) \\
final(a, (|_\circ, E_{init}, \bot, \bot)) &\triangleq final(init(a.l)) \vee final(init(a.r)) \\
final(a, (|_\circ, E, \mathsf{left}, s)) &\triangleq final(a.l, s) \\
final(a, (|_\circ, E, \mathsf{right}, s)) &\triangleq final(a.r, s)
\end{aligned}
$$

2.2.5.2 Semantics

There are four rules of inference. The first two deal with the execution of the first event from the initial state.

$$
|_1 \frac{init(a.l) \xrightarrow{\sigma, E_g, E''_g}_{a.l} s' \qquad \Theta}{(|_\circ, E_{init}, \bot, \bot) \xrightarrow{\sigma, E_e, E'_e}_{a} (|_\circ, E', \mathsf{left}, s')}
$$

$$
|_2 \frac{init(a.r) \xrightarrow{\sigma, E_g, E''_g}_{a.r} s' \qquad \Theta}{(|_\circ, E_{init}, \bot, \bot) \xrightarrow{\sigma, E_e, E'_e}_{a} (|_\circ, E', \mathsf{right}, s')}
$$

The other one deal with execution of the subsequent events from the chosen sub-ASTD.

$$
|_3 \frac{s \xrightarrow{\sigma, \Gamma, E_g, E''_g}_{a.l} s' \qquad \Theta}{(|_\circ, E, \mathsf{left}, s) \xrightarrow{\sigma, E_e, E'_e}_{a} (|_\circ, E', \mathsf{left}, s')}
$$

$$
|_4 \frac{s \xrightarrow{\sigma, \Gamma, E_g, E''_g}_{a.r} s' \qquad \Theta}{(|_\circ, E, \mathsf{right}, s) \xrightarrow{\sigma, E_e, E'_e}_{a} (|_\circ, E', \mathsf{right}, s')}
$$

2.2.6 Quantified Synchronization

The quantified synchronization allows for the modeling of an arbitrary number of instances of an ASTD which are executing in parallel, synchronizing on events from Δ [310].

2.2.6.1 Syntax

A quantified synchronization ASTD is a structure $\langle |[]|:, x, T, \Delta, b\rangle$ where $x \in \mathsf{Var}$ a quantified variable that can be only accessed in read-only mode, T the type of x, $\Delta \subseteq \mathsf{Label}$ a synchronization set of event labels and $b \in \mathsf{ASTD}$ the body of the synchronization. The state of a quantified synchronization is of type $\langle |[]|:_\circ, E, f\rangle$ where $|[]|:_\circ$ is the constructor, E the values of attributes and $f \in T \to \mathsf{States}$ is a function which associates a state of b to each value of T. Initial and final states are defined as follows. Let a be a quantified synchronized ASTD.

$$\begin{aligned} \mathit{init}(a) &\triangleq (|[]|:_\circ, a.E_{\mathit{init}}, T \times \{\mathit{init}(a.b)\}) \\ \mathit{final}(a, (|[]|:_\circ, E, f)) &\triangleq \forall\, c : T \cdot \mathit{final}(a.b, f(c)) \end{aligned}$$

2.2.6.2 Semantics

Rule $|[]|:_1$ describe execution of events with no synchronization. Symbol c denotes the element of T chosen for the execution.

$$|[]|:_1 \frac{\alpha(\sigma) \notin \Delta \qquad f(c) \xrightarrow{\sigma, E_g \lhd \{x \mapsto c\}, E''_g}_{a.b} s' \qquad \Theta}{(|[]|:_\circ, E, f) \xrightarrow{\sigma, E_e, E'_e}_a (|[]|:_\circ, E', f \lhd \{c \mapsto s'\})}$$

Rule $|[]|:_2$ describe execution of an event with synchronization. All elements of T must execute σ, in any order. It generalizes rule $|[]|_3$ and requires commutativity.

$$|[]|:_2 \frac{\alpha(\sigma) \in \Delta \qquad \Omega_{qsyn} \qquad \Theta}{(|[]|:_\circ, E, f) \xrightarrow{\sigma, E_e, E'_e}_a (|[]|:_\circ, E', f')}$$

Premiss Ω_{qsyn} formalizes commutativity by universally quantifying over all permutations p of T (noted $p \in \pi(T)$) and using Es as a sequence of environments

storing the intermediate results of the computation of E_g'' from E_g by iterating over the elements $p(i)$ of p. Let $k = |T|$.

$$\Omega_{qsyn} \triangleq \left(\begin{array}{l} \forall\, p \in \pi(T) \cdot \exists\, Es \in 0..k \to \mathsf{Env} \wedge Es(0) = E_g \\ \wedge\ E(k) = E_g'' \wedge \forall\, i \in 1..k \cdot (\\ f(p(i)) \xrightarrow{\sigma, Es(i-1) \lhd\!+ \{x \mapsto p(i)\}, Es(i)}_{a.b} f'(p(i))) \end{array} \right)$$

2.2.7 Quantified choice

The quantified choice is very similar to an existential quantification in first-order logic. It allows for picking a value from a set and execute a sub-ASTD with that value [310]. The scope of the quantified variable is the sub-ASTD.

2.2.7.1 Syntax

A quantified choice ASTD subtype has the following structure:

$$\mathsf{QChoice} \triangleq \langle |\ :, x, T, b \rangle$$

where $x \in \mathsf{Var}$ denotes a quantification variable, T is a type and $b \in \mathsf{ASTD}$ is the quantified ASTD. The type of a quantification choice state is $\langle |\ :_\circ, [\bot \mid c], E, [\bot \mid s] \rangle$ where $|\ :_\circ$ is the constructor of the quantification choice state, $\bot$ is a constant indicating that the choice has not been made yet, $c \in \mathsf{Term}$ denotes the current value of the choice quantified variable once the ASTD choice has been made, E the values of attributes and $s \in \mathsf{States}$. Initial and final states are defined as follows. Let a be a quantified choice ASTD.

$$\begin{array}{rcl} \mathit{init}(a) & \triangleq & (|\ :_\circ, \bot, a.E_{init}, \bot) \\ \mathit{final}(a, (|\ :_\circ, \bot, E_{init}, \bot)) & \triangleq & \exists\, x : T \cdot \mathit{final}(a.b, \mathit{init}(a.b)) \\ c \neq \bot \Rightarrow \Big(\mathit{final}(a, (|\ :_\circ, c, E, s)) & \triangleq & \mathit{final}(a.b, s)\{x \mapsto c\}\Big) \end{array}$$

2.2.7.2 Semantics

There are two inference rules. They use the notion of environment to manage the quantification. When a transition is computed using rules, the value c bound to the quantification variable x is added to the execution environment (the one appearing on the transition arrow) and can be used to make the proof, in particular to check that the event received σ matches the transition event σ', after the environment has been applied as a substitution. This behavior is expressed hereafter.

$$|:_1 \frac{init(a.b) \xrightarrow{\sigma, E_g \Leftarrow \{x \mapsto c\}, E''_g}_{a.b} s' \qquad \Theta \qquad c \in T}{(|:_\circ, \bot, E_{init}, \bot) \xrightarrow{\sigma, E_e, E'_e}_a (|:_\circ, c, E', s')}$$

$$|:_2 \frac{s \xrightarrow{\sigma, E_g \Leftarrow \{x \mapsto c\}, E''_g}_{a.b} s' \qquad \Theta \qquad c \neq \bot}{(|:_\circ, c, E, s) \xrightarrow{\sigma, E_e, E'_e}_a (|:_\circ, c, E', s')}$$

2.2.8 Kleene

This operator comes from regular expressions. It allows for iteration on an ASTD an arbitrary number of times (including zero). When the sub-ASTD is in a final state, it enables to start a new iteration. A Kleene closure is in a final state when it has not started or when its sub-ASTD is in a final state [310].

2.2.8.1 Syntax

The kleene ASTD subtype has the following structure:

$$\mathsf{Closure} \triangleq \langle \star, b \rangle$$

where $b \in \mathsf{ASTD}$ is the body of the closure. The type of a closure state is $\langle \star_\circ, E, started?, s \rangle$ where $s \in \mathsf{States}$, E the attribute values of the closure ASTD and $started? \in \mathsf{Boolean}$ indicates whether the first iteration has been started. It is essentially used to determine if the closure can immediately exit (i.e., is in a final state) without any iteration. Initial and final states are defined as follows. Let a be a closure ASTD.

$$\begin{aligned} init(a) &\triangleq (\star_\circ, a.E_{init}, \mathsf{false}, \bot) \\ final(a, (\star_\circ, E, started?, s)) &\triangleq final(a.b, s) \vee \neg started? \end{aligned}$$

2.2.8.2 Semantics

There are two inference rules: $\star_1$ allows for restarting from the initial state of the sub-ASTD when a final state has been reached; $\star_2$ allows for execution on the sub-ASTD.

$$\star_1 \frac{final(a.b, s)([\![E_g]\!]) \qquad init(a.b) \xrightarrow{\sigma, E_g, E''_g}_{a.b} s' \qquad \Theta}{(\star_\circ, E, \_, s) \xrightarrow{\sigma, E_e, E'_e}_{a} (\star_\circ, E', \mathsf{true}, s')}$$

$$\star_2 \frac{s \xrightarrow{\sigma, E_g, E''_g}_{a.b} s' \qquad \Theta}{(\star_\circ, E, \_, s) \xrightarrow{\sigma, E_e, E'_e}_{a} (\star_\circ, E', \mathsf{true}, s')}$$

2.2.9 Guard

The guard ASTD is a generalization of the guard specified on an automaton transition [310]. It is especially useful when the sub-ASTD is a complex structure, avoiding the duplication of the guard predicate on all the possible first transitions of that structure.

2.2.9.1 Syntax

The guard ASTD subtype has the following structure:

$$\mathsf{Guard} \triangleq \langle \Rightarrow, g, b \rangle$$

where $b \in \mathsf{ASTD}$ is the body of the guard. The type of a guard state is $\langle \Rightarrow_\circ, E, started?, s \rangle$ where $started?$ denotes when the guard has been satisfied, $s \in \mathsf{States}$ and E the attribute values of the guard ASTD. Initial and final states are defined as follows. Let a be a guard ASTD.

$$\begin{aligned} init(a) &\triangleq (\Rightarrow_\circ, a.E_{init}, \mathsf{false}, init(a.b)) \\ final(a, (\Rightarrow_\circ, E_{init}, \mathsf{false}, init(a.b))) &\triangleq final(a, init(a.b)) \\ final(a, (\Rightarrow_\circ, E, \mathsf{true}, s)) &\triangleq final(a, s) \end{aligned}$$

2.2.9.2 Semantics

There are two inference rules: $\Rightarrow_1$ deals with the first transition and the satisfaction of the guard predicate; $\Rightarrow_2$ deals with subsequent transitions.

$$\Rightarrow_1 \frac{g([\![E_g]\!]) \qquad init(a.b) \xrightarrow{\sigma,E_g,E''_g}_{a.b} s'}{(\Rightarrow_\circ, E_{init}, \mathsf{false}, init(a.b)) \xrightarrow{\sigma,E_e,E'_e} (\Rightarrow_\circ, E', \mathsf{true}, s')}$$

$$\Rightarrow_2 \frac{s \xrightarrow{\sigma,E_g,E''_g}_{a.b} s'}{(\Rightarrow_\circ, E, \mathsf{true}, s) \xrightarrow{\sigma,E_e,E'_e} (\Rightarrow_\circ, E', \mathsf{true}, s')}$$

2.2.10 Call

It is possible to call an ASTD which is defined in another diagram. A call is graphically represented by the called ASTD name and its actual parameter values [310]. Calls can be recursive.

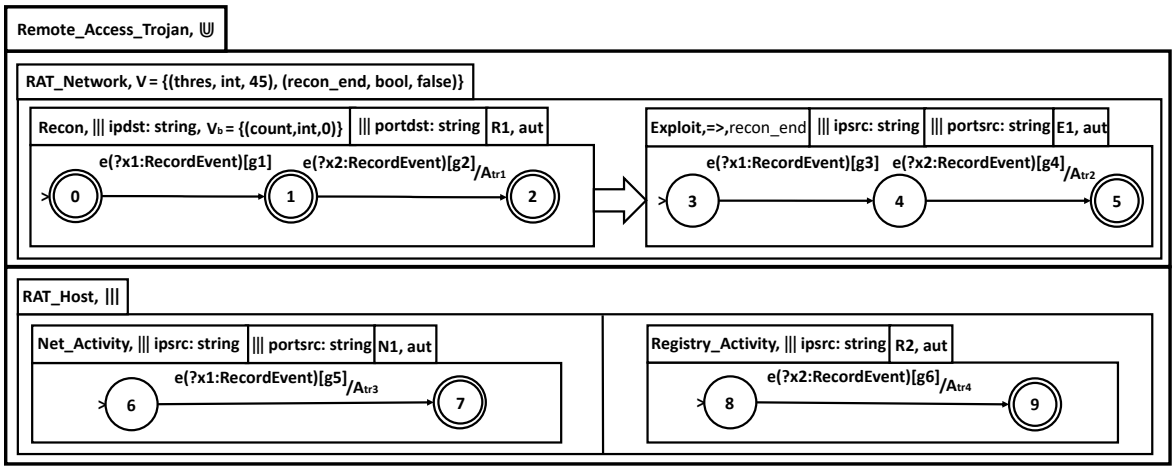
2.2.10.1 Syntax

A call ASTD is of the subtype $\mathsf{ASTDCall} \triangleq \langle \mathsf{cal}, n(\vec{c}) \rangle$ where n is the name of an ASTD $q = \langle n, P, V, A_{astd} \rangle$. Let $P = \vec{x} : \vec{T}$. For each $c_i \in \vec{c}$, we have $c_i \in T_i$. The type of an ASTD call state is $\langle \mathsf{cal}_\circ, E, [\bot \mid s] \rangle$, where $\mathsf{cal}_\circ$ is the constructor of the call state, E the values of attributes, $\bot$ denotes that the call has not been made yet and $s \in \mathsf{States}$ is the state of the called ASTD q once the called has been made. The initial and final states are as follows. Let a be an ASTD call.

$$\begin{aligned} init(a) &\triangleq (\mathsf{cal}_\circ, a.E_{init}, \bot) \\ final(a, (\mathsf{cal}_\circ, E_{init}, \bot)) &\triangleq final(q, init(q))([\vec{x} := \vec{c}]) \\ s \neq \bot \Rightarrow final(a, (\mathsf{cal}_\circ, E, s)) &\triangleq final(q, s)([\vec{x} := \vec{c}]) \end{aligned}$$

2.2.10.2 Semantics

There are two rules of inference. Rule cal_1 deals with the initial call execution, while cal_2 deals with subsequent executions. Let $E_{cal} = \{P \mapsto \vec{c}\}$.



g1 = (x1.ipdst=ipdst) && (x1.portdst=portdst) && (x1.proto="TCP") && (x1.tcpflags= "S") && (**not** recon_end)
g2 = (x2.ipsrc=ipdst) && (x2.portsrc=portdst) && (x2.proto="TCP") && (x2.tcpflags= "RA")
g3 = (x1.ipsrc=ipsrc) && (x1.portsrc=portsrc) && (x1.proto="TCP") && (x1.tcpflags= "S")
g4 = (x2.ipdst=ipsrc) && (x2.portdst=portsrc) && (x2.proto="TCP") && (x2.tcpflags= "SA") && (contains x2.payload "stdapi")
g5 = (x1.ipsrc=ipsrc) && (x1.portsrc=portsrc) && (x1.proto="TCP") && (x1.eventid= "3")
&& (**List.mem** x1.portdst ["80"; "443"; "444"; "445"; "4444"]))
g6 = (x2.ipsrc=ipsrc) && (x2.eventid= "13") && (contains x2.registrykey "HK\\(U\\|LV\\|LM\\|CR\\)\\\\") &&
(contains x2.registrykey "\\\\Run\\\\[a-zA-Z]+ .\\(exe\\|vbs\\|bat\\|lnk\\)")

Atr1 = { count := !count+1;
if !count >= !thres **then**
alert "Port scan attack";
count := 0; recon_end := **true**;;}

Atr2 = {alert "Metasploit privilege escalation";;}

Atr3 = {alert "File ' "^x1.image^" ' attempted a suspicious connection to "^x1.ipdst^" on port "
^x1.portdst;;}

Atr4 = {alert "Malware registry attack";;}

Figure 2.4 – Remote Access Trojan ASTD specification

$$\mathsf{cal}_1\ \frac{init(a.b) \xrightarrow{\sigma, E_g \lhd E_{cal}, E''_g}_{a.b} s' \qquad \Theta}{(\mathsf{cal}_\circ, E_{init}, \bot) \xrightarrow{\sigma, E_e, E'_e}_{a} (\mathsf{cal}_\circ, E', s')}$$

$$\mathsf{cal}_2\ \frac{s \xrightarrow{\sigma, E_g \lhd E_{cal}, E''_g}_{a.b} s' \qquad \Theta}{(\mathsf{cal}_\circ, E, s) \xrightarrow{\sigma, E_e, E'_e}_{a} (\mathsf{cal}_\circ, E', s')}$$

2.3 Case Study

Fig. 2.4 illustrates a case study of ASTDs in cyber attack detection. Increasingly, attackers develop various strategies to break down existing defence systems and, consequently, gain unauthorized access to a private Information System (IS). They operate strategically by executing a sequence of threatening actions [2] to dis-

2. https://attack.mitre.org/wiki/All$_T echniques$

cover the network topology and vulnerabilities on the target IS, followed by a phase exploiting the vulnerabilities found to command and control the target. We illustrate this approach by an active attack called Remote Access Trojan (RAT) [123]. A RAT attack operates both on a network and a host. It starts on a network through vulnerability scans. Next, the attacker sends a malware to the victim machine (e.g., using email spams) for exploitation. Once the malware is installed on the victim's machine, it tries to automatically connect to the attacker machine, and an attack session is opened when user clicks on the infected program (exploitation). Then, events are generated from both host and network sides. Considering both gives better insights (or a holistic view) of RAT activities.

The main ASTD is identified by the attack name Remote_Access_Trojan in the tab of the box. The name can be omitted for nested ASTDs. Remote_Access_Trojan illustrates the extension: it declares two attributes, thres and recon_end, with their types and initial values. Remote_Access_Trojan is a flow ASTD (denoted by ⋓) that concurrently executes events from two attack models in a network and host: RAT_Network and RAT_Host. RAT_Network allows for sequential composition of two ASTDs: Recon and Exploit. The first ASTD of the sequence (i.e., Recon) must reach a final state before the next one can start.

The ASTD Recon starts its execution and inspects the network traffic to detect port scanning and operating system (OS) detection attempts. Attempts may be done by an attacker who tries to scan open ports and OS vulnerabilities (e.g., system errors, bugs) on the target IS (victim). Attacker actions generate network traffic that contains malicious patterns enabling identifying the current attack. Recon is a quantified interleaving ASTD (denoted by ||| ipdst : string) that allows an arbitrary number of instances of the nested ASTD to be executed in interleaving, each instance being indexed by its ip address ipdst. It also declares an attribute count whose value is shared by all its interleave instances. Note that attributes can be declared within any ASTD. ASTD Recon has a nested nameless ASTD which is a quantified interleaving on the destination port (i.e., ||| portdst : string). For each possible value taken by ipdst and portdst, the nested automaton R1 tracks the scanning of a port; it also has read-write access of the attributes count and thres. However, R1 can not modify ipdst and portdst, because quantified interleave variables are read-only. Being called within two

quantified interleaves, there is an instance of this automaton for each pair of values of ipdst and portdst. Its initial state is 0, depicted by ⤚○ and it is also final (denoted by ◎). This state has an outgoing transition labeled by the event e(?x1:RecordEvent) and a guard [g1]. Variable x1 is a local variable whose scope is the transition only. Type RecordEvent is a record containing both host and network events in the simplified form

$\langle systime, eventid, image, registrykey, proto, ipsrc, portsrc, ipdst, portdst, tcpflags, payload \rangle$

where *systime* is the date-time when the event occurred, *eventid* is the event ID [3] (e.g., "1"=Process Creation, "3"=Network Connection, "13"=Registry Value Set), *image* the file path of the running process, *registrykey* the path of the registry object, *proto* is the protocol type (e.g., ICMP, TCP, UDP), *ipsrc* the source address, *portsrc* the source port, *ipdst* the destination address, *portdst* the destination port, *flags* a combination of TCP flags (S/SYN initiates a connection, A/ACK acknowledges received data, R/RST aborts a connection in response to ans error), and *payload* the event content.

The guard states that the source initiates a TCP connection to the destination, the destination ip address and port of the event matches the quantified interleave variables ipdst and portdst, and the end of the reconnaissance phase has not been reached. This means that the event is executed only on the appropriate automaton instance. The transition from state 1 to state 2 captures the response of the victim to the attacker. It illustrates the declaration of actions on transitions, the second type of extensions made to the ASTD notation. Action A_{tr1} increases attribute count of Recon. When count reaches the threshold, attribute recon_end is set to true, which will enable ASTD Exploit to start. Actions are expressed in OCaml to integrate easily with the ASTD interpreter. An attribute x is represented by an OCaml reference variable x, which is dereferenced using "!x" to access its value.

Let consider the following events which are used to describe the RAT specification.

3. https://docs.microsoft.com/en-us/sysinternals/downloads/sysmon

```
revt1 <systime="t1", eventid="", image="", registrykey="", proto="TCP", ipsrc="ip1",
portsrc="pt1", ipdst="ip2", portdst="pt2", tcpflags="S", payload="p1">

revt2 <systime="t2", eventid="", image="", registrykey="", proto="TCP", ipsrc="ip2",
portsrc="pt2", ipdst="ip1", portdst="pt1", tcpflags="RA", payload="p2">

revt3 <systime="t3", eventid="", image="", registrykey="", proto="TCP", ipsrc="ip1",
portsrc="pt1", ipdst="ip2", portdst="pt3", tcpflags="S", payload="p3">

revt4 <systime="t4", eventid="", image="", registrykey="", proto="TCP", ipsrc="ip2",
portsrc="pt3", ipdst="ip1", portdst="pt1", tcpflags="RA", payload="p4">

revt5 <systime="t5",eventid="3",image="C:\\KWIAYA\\XYkgwUUo.exe", registrykey="",
proto="TCP", ipsrc="ip2", portsrc="pt2", ipdst="ip1", portdst="pt1", tcpflags="S",
payload="p5">

revt6 <systime="t6", eventid="", image="", registrykey="", proto="TCP", ipsrc="ip1",
portsrc="pt1", ipdst="ip2", portdst="pt2", tcpflags="SA", payload="stdapi">

revt7 <systime="t7",eventid="13",image="C:\\ProgramData\\pykEMEsI\\ykAYMkMA.exe",
registrykey="HKLM\\SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Run\\EUgUgAYs.exe",
proto="UDP", ipsrc="ip1", portsrc="pt4", ipdst="ip1", portdst="pt5", tcpflags="",
payload="">
```

An attacker initiates a TCP connection on its machine (address ip1) to the victim machine (address ip2). The reception of the event e(revt1) triggers a transition from 0 to 1, within Recon in the interleavings instance ip2 and pt2. The state 2 is reached when event revt2 is received. Action A_{tr1} increments attribute count and returns an immediate alert when the number of scanned ports on the victim machine reaches a threshold. An alert is a message sent to the environment (e.g., stdout). In this case, the attribute recon_end takes the value true, enabling the next component of the sequence (i.e., Exploit) to start. After receiving event revt4, the value of count is 2 and two instances of R1 (i.e., (ip2,pt2) and (ip2,pt3) are in state 2; the others are still in their initial state).

Because ASTD Recon is a quantified interleave, it is final when all its interleaved instances are final; similarly for its nested ASTD ||| portdst : string. An automaton is final when its current state is final. Since all the states of R1 are final, then ASTD Recon is always final. Thus, ASTD Exploit is always enabled, but it is a guard ASTD, identified by the operator $\Rightarrow$. It can start only if its guard condition (recon_end) is true. The introduction of attributes allows us to define more specific guard conditions that can be used in guarded ASTDs in a sequential composition. It is an alternative way of controlling the sequential composition of ASTDs: since each automaton state

is final, it is the guard that decides when the second component of a sequence can trigger.

The exploit phase starts when the victim machine runs an infected program (weapon) that communicates with the attacker machine. It allows the attacker to send malicious payloads to the victim by exploiting the Server Message Block (SMB) vulnerability [4]. Payloads contain standard remote call API (STD API) signatures that enable detecting attacker activities. The nested ASTDs of Exploit respectively interleave on the source address (i.e., ||| ipsrc : string) and the destination address (i.e., ||| ipdst : string). This allows a number of instances of E1 for each possible of ipsrc and ipdst. The automaton ASTD E1 is in state 3.

On the reception of an event e(revt5), transitions from 3 to 4 and 6 to 7 are synchronously executed by the flow operator. It means that the victim has clicked on the malware file and exploit succeeded. In the host, the malware attempts a network connection (i.e., revt5.eventid="3") to the attacker machine on port 4444. The function List.mem elt list returns true when elt exists in list. In the network, an attack session has been initiated (i.e., revt5.flags="S") between the victim and the attacker. The next transition (i.e., from 4 to 5) is executed when an attack session is established (i.e., revt6.flags="SA") and the attacker starts to send malicious payloads (e.g., "stdapi" in the revt6 payload). Concurrently, the malware attempts to locally modify registries (i.e., revt7.eventid="13") on the victim machine ip1. Its behaviour is identified by a regular expression in the registry key [5]. The function contains str regexp returns true, when a string in str matches the regular expression regexp.

2.4 ASTD Tool Support

2.4.1 Prolog and ProB

The ProB model checker was originally developed for B specifications, but can also be applied to other languages whose operational semantics are expressed in Prolog. This is how the synchronization between CSP and B was realized in [41], by

4. https://cert.europa.eu/static/SecurityAdvisories/2017/CERT-EU-SA2017-012.pdf
5. https://support.microsoft.com/en-ca/help/256986/windows-registry-information-for-advanced-users

transcribing the operational semantics of CSP to Prolog. We have implemented the operational semantics of extended ASTDs along with all features and operators as required for this case study: sequence, guard, and various synchronization operators (interleaving, full and selective synchronization) and the new flow operator.

The operational semantics rules of extended ASTDs can be translated to individual Prolog clauses. For example, the rule for the sequence operator is translated as follows.

```
atrans(seq([A1|T]),G1,Trans,S2,G2) :- aseq_trans(A1,T,
G1,Trans,S2,G2).
aseq_trans(A1,T,G1,Trans,S2,G2) :- %allow A1 to evolve
atrans(A1,G1,Trans,A2,G2), create_seq(A2,T,S2).
aseq_trans(A1,[A2|T],G1,Trans,S2,G2) :-
% if A1 is final: skip to rest
is_final_astd(A1), aseq_trans(A2,T,G1,Trans,S2,G2).
```

Intuitively, the Prolog predicate `atrans(S1,G1,E,S2,G2)` is true when the ASTD expression S1 can execute the event E in the context of the global state G1, resulting in a new ASTD expression S2 and a new global state G2. In other words, it corresponds to $S1 \xrightarrow{E,G1,G2}_a S2$ from Sect. 2.2. The Prolog predicate `is_final_astd(A)` is true when the ASTD A is final, implementing *final* from Sect. 2.2. For efficiency reasons, `atrans` calls other subsidiary predicates like `aseq_trans` and `create_seq`. The latter simply constructs a new ASTD sequence expression.

An ASTD expression is a Prolog term representing the ASTD structure, e.g., `seq([aut(R1 ('8.8.8.8',80), aut(E1('8.8.8.8',80)])` to represent the sequential composition of an instance of the R1 automata and an instance of the E1 automata from Fig. 2.4. The global state is represented as a list of bindings, e.g., `[count/0,recon_end/0,thres/45, warnings/[]]` for R1 in Fig. 2.4.

Note that here we have implemented sequence not as a binary operator like in Sect. 2.2.2, but also as n-ary operator that combines several ASTDs. The first clause of `aseq_trans` corresponds to the rule $⇨_1$ from Sect. 2.2.2, and the second clause to rule $⇨_2$. There is no need for rule $⇨_3$ as we throw away A1 in the resulting process expression S2.

Currently, to run ProB on an ASTD, one needs to express the transitions of the individual automata in Prolog. For normal automata, this results in one fact

per transition. For automata with actions and rules, this corresponds to one Prolog clause per transition, the body of the clause containing the conditions and associated actions. In future, we plan to generate those Prolog translations automatically from the ASTD representation.

By writing the interpreter, we have gained access to various features of ProB: animation, model checking (deadlock, determinism, safety, LTL, CTL), refinement checking, and execution (a faster version of animation which does not store the history of states). In principle one could also synchronize ASTDs with B machines in the style of [41].

As a first simpler example we have replicated the Library case study from [96], using global state to store reservations. Our Prolog ASTD interpreter is about three times faster than ProB on the B translation of the system (not using ProB's symmetry reduction or partial order reduction). On our security case study, the animation features uncovered various issues with earlier versions of the attack models of Fig. 2.4. For example, the animator and model checker uncovered various unexpected non-determinisms in the attack specification. We have also managed to replay a real attack log, validating that attacks are indeed correctly detected. For this experiment, we have written an ASTD which reads in a log file and replays the events in the file. This ASTD is put into parallel with the attack model, and one can check whether attacks are identified or not. The execution took 50 ms to process 300 events.

2.4.2 ASTD OCaml Interpreter

In [279], an initial development of an ASTD interpreter has been made for information systems. It has been extended to support new features for cyber attack detection. The interpreter implements the operational semantics of ASTDs by computing transition proofs. It executes an input attack specification on input sources (i.e., packets, audit logs). Attack specifications are converted into a serialized format [310] before being sent to the interpreter. During execution, raw events from the host or/and network are preprocessed into the enumerated form e(systime, eventid, image, ...). The interpreter reads preprocessed events in offline mode (i.e., from a file) or in real-time, and computes possible transitions of the specification. Actions

like alerts are displayed to the administrator. Hereafter, some alerts generated from the *i*ASTD output during detection.

```
Alert Port scan attack
Alert Metasploit privilege escalation

Alert File 'C:\\Users\\admin\\Desktop\\ZisUurWz.exe' attempted a suspi
cious connection to 192.168.1.129 on port 4444

Alert File 'C:\\ProgramData\\JAsIssEI\\EUgUgAYs.exe' attempted a suspi
cious connection to 172.217.3.206 on port 80
```

Several attacks like RAT, Ransomwares, and Lateral Movement have been specified and executed using the interpreter [6].

2.5 Discussion and Conclusion

This paper proposes an extension of the ASTD notation with attributes, actions and a new operator called flow. These extensions are particularly useful to model cybersecurity attacks and are implemented in two interpreters, one in Prolog with ProB, and the other in OCaml. ProB proved to be a useful addition, because it gives us access to several model checking features already implemented, like refinement checking, determinacy checking, temporal formula checking. However, these interpreters are currently limited to primitive types integers and strings.

The algebraic nature of our approach is quite useful to explore several variants of specification attacks. For instance, the specification of Fig. 2.4 triggers the exploit phase detection when one ip address has been scanned. An alternative behaviour is to trigger one exploit phase for each ip address scanned; it suffices to move the sequence operator just after ASTD Recon, and remove the quantified interleave ||| portsrc: string in ASTD Exploit. Another variant would be to use a synchronization ||| instead of a sequence in RAT_Network; this would enable to detect both port scans and exploits concurrently. Making these changes in a model-based language like B, or in a scripting language like Python, which is the common practice in security, would entail several changes in the types of variables and the modification of preconditions and

6. The interpreter and results are available at https://depot.gril.usherbrooke.ca/fram1801/iASTD-public.

postconditions in several operations, which is subtle and error-prone. The ability to compose specifications relieves the specifier from this burden. The ASTD language is accessible by users who are not necessarily experts and the attack detection is automatically done by the interpreters. A script that reproduces the same behaviour for attack detection, requires a huge case analysis on the content of the packet and audit logs, something which is hard to code and maintain.

Our future work consists in building the ability to define complex types (e.g., Packet, Flow, Session, AuditLog) to handle the detection of more complex attacks. We plan to use ontologies to define these types and use them in various tools to support our approach. Further experimentation will be conducted to handle more complex attack specifications. If these are successful, we plan to develop an ASTD compiler that will generate efficient code from ASTD specifications and use them to detect intrusion in real environments. Quantified interleaves can be efficiently executed in constant time or $O(\log(n))$ when the quantification variable occurs in each event, which is typically the case [94].

Chapitre 3

Détection d'intrusions avec les ASTDs

Résumé

Dans cet article, nous montrons l'application des ASTD à la détection d'intrusion. ASTD est une notation exécutable, modulaire et graphique qui permet la composition de machines à états hiérarchiques avec des opérateurs d'algèbre de processus pour modéliser des phases d'attaque complexes. Dans l'ensemble, les spécifications d'attaque ASTD sont plus concises que celles écrites dans le langage de Snort, Zeek et d'autres langages de la littérature. Pour la détection d'intrusion, *i*ASTD (l'interpréteur ASTD) et Zeek ont fourni des résultats similaires. *i*ASTD a produit moins de faux positifs et un plus petit nombre de vrais positifs par attaque que Snort, ce qui est un facteur important pour gérer d'énormes quantités d'événements. Le temps de traitement de *i*ASTD sur le banc de test en temps réel est plus lent que Snort et Zeek, mais il peut être amélioré en compilant les spécifications ASTD dans des scripts Zeek.

Commentaires

La contribution ici réside dans l'élaboration d'une méthodologie de spécification d'attaques avec la notation étendue des ASTDs et la comparaison de l'outil implémentant le langage avec des outils industriels tels que Snort et Zeek. La notation étendue est plus modulaire, concise et réutilisable que celles répertoriées dans la littérature. L'évaluation des outils est effectuée en temps réel et sur des ensembles de données existants. Plus de 65 attaques ont été simulées et spécifiées avec la notation en collaboration avec Nokia Canada. Une vingtaine d'attaques a été spécifiée à partir d'énormes ensembles de données réelles provenant du Centre de la Sécurité des Télécommunications (CSE-CIC-IDS2018) et de l'Université Technique Tchèque de Prague (CTU). Pour la corrélation d'événements, l'outil implémentant l'approche a produit peu de fausses alertes et est plus précis que Snort et Zeek [312].

Les contributions décrites dans ce chapitre ont fait l'objet d'un article publié dans le cadre de la $34^{ième}$ édition de la conférence internationale *AINA (International Conference on Advanced Information Networking and Applications)*, de rang B, qui a eu lieu à Caserta, Italie, le 15 avril 2020.

Les contributions et l'article sus-cité ont été élaborées par mes soins en tenant compte des remarques et commentaires issus de mon équipe d'encadrement.

Intrusion Detection using ASTDs

Lionel N. Tidjon
Université de Sherbrooke, GRIF, Québec, Canada
Télécom SudParis, SAMOVAR, Institut Polytechnique Paris, France
lionel.nganyewou.tidjon@usherbrooke.ca
Marc Frappier
Université de Sherbrooke, GRIF, Québec, Canada
marc.frappier@usherbrooke.ca
Amel Mammar
Télécom SudParis, SAMOVAR, Institut Polytechnique Paris, France
amel.mammar@telecom-sudparis.eu

Keywords: Formal Specification, ASTD, Intrusion Detection

Abstract

In this paper, we show the application of ASTDs to intrusion detection. ASTD is an executable, modular and graphical notation that allows for the composition of hierarchical state machines with process algebra operators to model complex attack phases. Overall, ASTD attack specifications are more concise than industrial tools like Snort, Zeek, and other attack languages in the literature. For intrusion detection, *i*ASTD (the ASTD interpreter) and Zeek provided similar results. *i*ASTD produced less false positives and a smaller number of true positives per attack than Snort, which is an important factor to deal with huge amounts of events. The processing time of *i*ASTD on the real-time testbed is slower than Snort and Zeek, but it can be improved by compiling ASTD specifications into Zeek scripts.

3.1 Introduction

In Security Operations Center (SOCs), several intrusion detection tools are placed at different levels of the network to ensure the security and privacy of information [311]. Snort [271], a widely used IDS, provides a low-level signature language to express and detect multi-stage Advanced Persistent Threats (APT) attacks. Zeek [246] was proposed to overcome some limitations of Snort by providing an event-driven scripting language to precisely specify and identify APT attacks. The writing of Zeek scripts is essentially programming using functions and global variables. Eckmann *et al.* [91] have proposed STATL, a stateful and domain-independent language that allows a more abstract representation of attack scenarios than Snort and Zeek using state machines with actions and state variables. Other attack languages like LAMBDA [66] and Chronicle [214] have been proposed. LAMBDA [66] provides an abstract description of an attack operation in terms of conditions and effects, expressed using predicates. Chronicle [214] reconstructs a state machine from event patterns that can be ordered with timing constraints. Barringer *et al.* [29] have introduced quantified event automata (QEA), in which universal and existential quantification are used to quantify parameters of an automaton, allowing to replicate an automaton and efficient execution.

In this paper, we show the application of ASTDs to intrusion detection. ASTD is an executable, modular and graphical notation that allows for the composition of hierarchical state machines using process algebra operators such as flow, sequence, quantified interleaving and parallel synchronization [97,234]. It allows one to capture "big picture" of complex attacks by graphically specifying their behaviours in a modular fashion, defining complex relationships between events (i.e., event correlation) to model and detect attack phases. ASTDs can be seen as extensions of STATL, since state machines are elementary ASTDs. STATL does not compose state machines using process operators; thus it is less modular. ASTDs offer a more abstract representation of attacks than LAMBDA, since the logical expressions of attacks are low-level mechanisms to deal with APT attacks. ASTD operators can be encoded into Chronicle, but at the expense of loosing abstraction and concision. Quantified versions of synchronization and interleaving ASTDs [234] are generalisations of QEA's

quantifications. These quantified versions provide the ability to replicate ASTDs and index them with a quantified variable (e.g., address, port), which is necessary to construct specifications which are more resilient to attack mutations and variants.

Our specification approach using ASTDs is based on attack pattern databases like MITRE's Common Attack Pattern Enumeration and Classification (CAPEC) [1] and ATT&CK (Adversarial Tactics, Techniques & Common Knowledge) [283]. We propose to specify a case study of ransomwares with different Snort, Zeek, and ASTD in the litterature to identify their weaknesses and strengths. The specification of recent malwares including ransomwares has been conducted in collaboration with Nokia Threat Intelligence Centre. The aim was to get feedbacks from cybersecurity experts and improve the ASTD language for intrusion detection.

Among existing tools, we have selected IDSs like Snort and Zeek for comparison; because they are widely used and well maintained by the cybersecurity community. Tools related to other attack languages in the literature were either no longer available, or deprecrated i.e. not able to run on current operating systems, or not working operationally on real world environments (essentially worked on old datasets). Our results show that the ASTD notation is more abstract, modular and concise than Snort and Zeek, while achieving good performance on heterogeneous event sources, thanks to its advanced event correlation capabilities [234]. Attack detection is done by executing ASTD specifications on online and offline events using the *i*ASTD tool, whose processing time is slower than Snort and Zeek, but it can be improved by compiling ASTD specifications into Zeek scripts or other programming languages.

The rest of this paper is structured as follows. The methodology for ASTD attack specification is described in Section 3.2. In Section 3.3, we present the specification of a case study using Snort, Zeek, and ASTD. Section 3.4 describes the execution of attack specifications by the tools. In Section 3.5, we compare and discuss the results of the *i*ASTD tool against Snort and Zeek. Section 3.6 concludes with some perspectives.

1. http://makingsecuritymeasurable.mitre.org/docs/capec-intro-handout.pdf

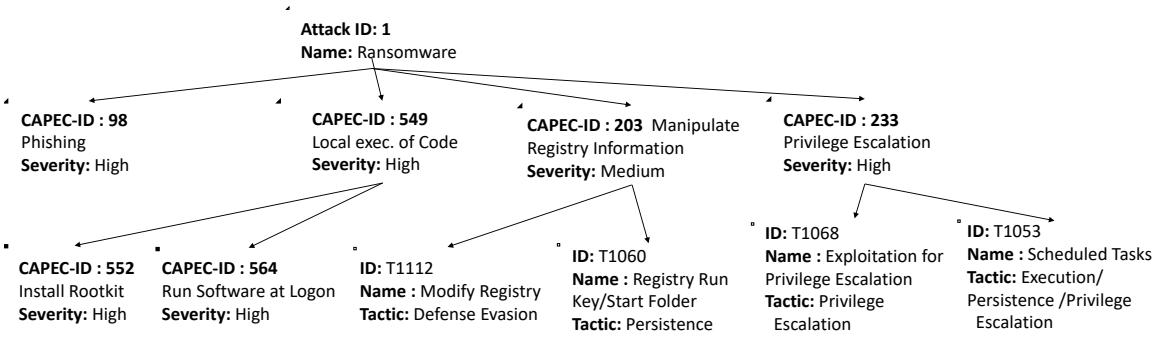


Figure 3.1 – Ransomware attack pattern from CAPEC and ATT&CK

3.2 Attack Specification Methodology using ASTDs

CAPEC[2] is one of the most well-known attack pattern database. Cybersecurity companies use it to figure out how cyber actors exploit weaknesses in applications and platforms. Another interesting database is ATT&CK [283], which complements CAPEC in providing more details about attacker actions and techniques. Attack patterns are hierarchical descriptions: they are decomposed into phases, which are further decomposed into steps. A step is realized using a combination of events. A phase or a step may appear in several attack patterns, so there is an interest in describing phases and steps independently and to compose them to build an attack specification.

The ASTD notation provides the necessary operators to construct modular formal models of attack patterns. Each phase and step can be defined by its own ASTD, properly encapsulated and parameterized, in order to allow its reuse in several attack patterns. Attack patterns can be composed together to create a global ASTD specification of an IDS.

To illustrate our approach, we show an attack pattern for ransomwares in Fig. 3.1, extracted from CAPEC and ATT&CK. The pattern states that the attacker starts by delivering an attached file to the victim machine through email phishing. The victim downloads the malware by clicking on the malicious link in the email (CAPEC-98). The malware locally executes and encrypts user files (CAPEC-549). It also modi-

2. http://makingsecuritymeasurable.mitre.org/docs/capec-intro-handout.pdf

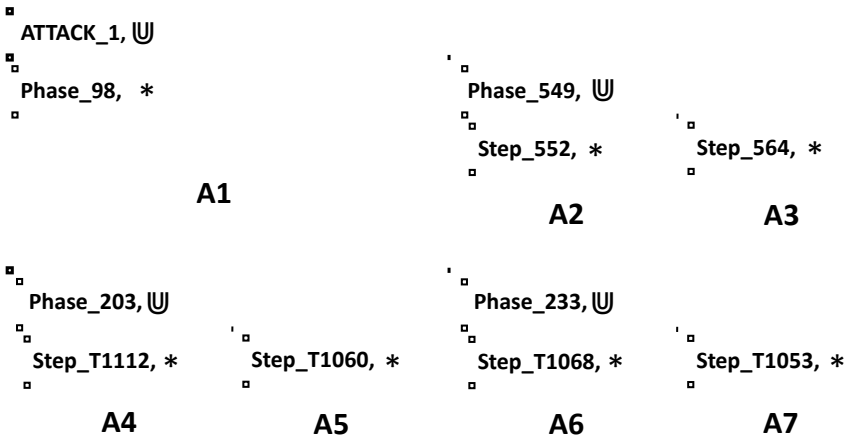


Figure 3.2 – ASTD specification of attack pattern of Fig. 3.1

fies the registry by adding an entry to the "run keys" in the registry to be persistent (CAPEC-203). Concurrently, the malware spreads itself and executes malicious scheduled tasks on the local or remote system (CAPEC-233).

Fig. 3.2 shows the top-level ASTD specification of the attack pattern of Fig. 3.1. Attack phases and steps are composed using the flow operator ⋓ to execute them in parallel. Phases are intuitively perceived as sequential, but in practice, they may overlap, thus their composition is better represented by a flow. The Kleene closure "$\star$" allows iterating on each attack step as the attacker can execute the same step several times. In each step, A_i ($i \in 1..7$) denotes a call to an ASTD which represents the ASTD step, typically in terms of an automaton.

3.3 Specification of a case study

We have specified more than 65 malware variants (including ransomwares) from Nokia, targeting different operating system (Windows, Linux, Android, iOS) and 20 other malware variants from theZoo github project using the ASTD, Zeek, and Snort languages. In this section, we specify a recent variant of ransomwares called Gandcrab using ASTD, Snort, and Zeek. The specification of this variant in other attack

languages like STATL can be found on Git [3]. Gandcrab can be resumed into 6 actions (see Appendix A.1). In action 1, the attacker delivers an email containing an embedded file. Once the victim runs the attached file, it downloads and executes Gandcrab (action 2). In action 3, Gandcrab gets the IP address of the victim host by sending a DNS request to the website *ipv4bot.whatismyipaddress.com*. In action 4, Gandcrab replicates by creating a malicious file in the AppData folder (e.g., *yxvace.exe*). This malicious file checks-in multiple command and control (C&C) sites using the command *nslookup site_name dns_server*. During C&C check-in, Gandcrab also sends a HTTP GET request to its C&C site (action 5). Next, Gandcrab encrypts collected data in action 3 and post it to the C&C server (action 6).

Actions 1 and 2 are done in phase CAPEC-98. The remainders are done in phase CAPEC-549. Hereafter, we specify the Gandcrab case study using the Snort, Zeek and ASTD languages in order to compare their weaknesses and strengths.

3.3.1 ASTD specification

As Gandcrab operates at both the host and the network levels, we build one attack model for each and compose them using the flow operator, following the specification methodology. In Fig. 3.3, the main ASTD *Gandcrab_Ransom* is of type flow and declares two variables *v1* and *v2*, each of type boolean. These variables can be modified by actions. *Gandcrab_Ransom* composes phases *Phishing* (CAPEC-98) and *Exec_Code* (CAPEC-549) using the flow operator ⩏.

The first phase (i.e., *Phishing*) is a quantified interleaving ASTD that allows detecting the attacker's action 2. *Phishing* and its nameless sub-ASTD (i.e., |||*portsrc :string*) interleave instances of its sub-ASTD, indexed by variables *ipsrc* (source address) and *portsrc* (source port). Its sub-ASTD is a nameless Kleene closure and it is identified by ⋆. It allows iterating on an ASTD call that refers to the ASTD definition of *Downld* described in Fig. 3.3. *Downld* is an ASTD automaton that receives parameters *ipsrc*, *portsrc*, *v1*, and *v2* from its calling ASTDs. Being called within two quantified interleaves, there is an instance of this automaton for each pair of values of *ipsrc* and *portsrc*. The initial state 0 has an outgoing transition labeled by the event

3. https://depot.gril.usherbrooke.ca/fram1801/iASTD-public

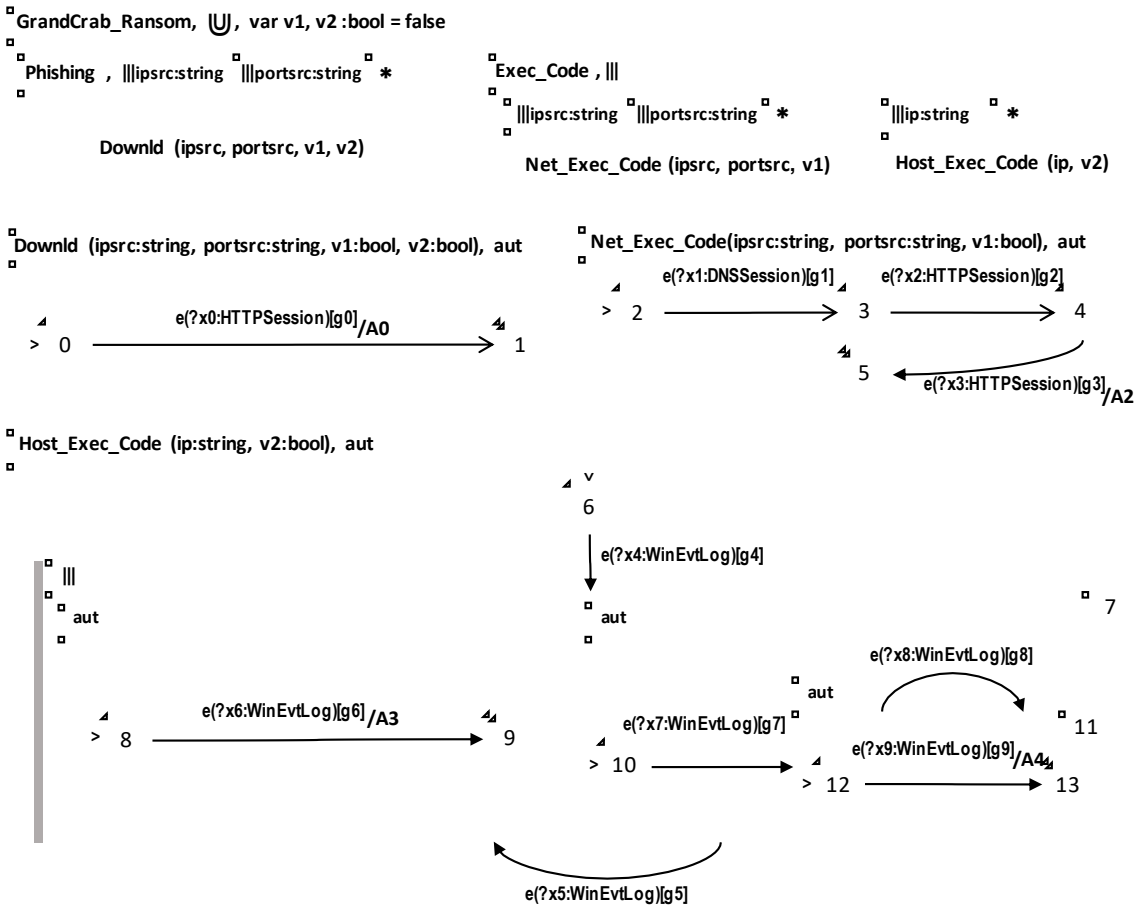


Figure 3.3 – Gandcrab crypto-worm specification

e(?x0:HTTPSession) and a guard *[g0]*. The local variable *x0* has a user-defined type *HTTPSession*. *HTTPSession* is described in an ontology to process multiple HTTP sessions from the network traffic.

From the initial state 0, the transition executes action *A0* (e.g., A0 = { v1 = true; v2 = true;}) when the guard *g0* is *true*. The guard *g0* checks if HTTP sessions contain signatures of the malicious file during downloading (e.g., GET /js/kukul.exe). When transition 0→1 is executed, *v1* and *v2* take value *true* to notify other ASTDs that the phishing phase is done. The second phase (i.e., *Exec_Code*) interleaves two nameless ASTDs. Each one respectively interleaves host and network events to identify Gandcrab actions. The first one (i.e., |||*ipsrc:string*) and its nested component (i.e., |||*portsrc: string*) allow multiple instances of the ASTD automaton *Net_Exec_Code*,

indexed by *ipsrc* and *portsrc*. Within *Net_Exec_Code*, the transition 2→3 tracks the Gandcrab action 3 in the network traffic. Next, the transition 3→4 detects the Gandcrab's action 5 when *g2* holds. From state 4, the transition 4→5 checks the malicious action 6. The action A2 (e.g., A2 = {**if** v1 **then** print "GandCrab CnC"; }) shows an alert only if the phishing phase tooks place.

Concurrently, the transition 6→7 in ASTD *Host_Exec_Code* also tracks the Gandcrab action 4 in host events. It is labeled by the event *e(?x4:WinEvtLog)* where *x4* is a transition local variable of type *WinEvtLog*. Type *WinEvtLog* has a structure similar to Windows event logs. The guard *g4* is *true* when the Gandcrab file creates a process that checks-in its C&C domains using the Windows command (e.g., *nslookup carder.bit ns1.wowservers.ru*). Once *g4* is *true*, the transition moves to the shallow final state 7. The state 7 is a complex state (i.e., an interleaving ASTD) that composes two ASTD automatons. Within the first automaton, the transition 8→9 checks if the previous nslookup command successfully established a DNS connection to C&C servers (e.g., ns1.wowservers.ru). Concurrently, the transition 10→11 detects when the nslookup command process forks into another process *svchost.exe* to leak system information over the open port 3389.

3.3.2 Snort specification

From the case study, we can deduce 4 detection signatures[4], each referring to phases *CAPEC-98* and *CAPEC-549* (see Appendix A.2). These signatures are low-level representations of transitions in ASTD automatons *Downld* and *Net_Exec_Code* (see Fig. 3.3). For example, the following signature

```
alert tcp $HOME_NET any -> $EXTERNAL_NET 80 (msg:"Malicious Software Downloading";
flow:established, from_client; content:"GET"; http_method; content:".exe HTTP/1.";
fast_pattern:only; content:"Connection: Keep-Alive"; http_header; content:"Accept
|3a 20|";http_header;content:"User-Agent: Mozilla";http_header;content: "Host|3a|";
pcre:"/Host\x3a\x20(:?[0-9]{1,3}\.){3}[0-9]{1,3}/H"; reference: capec,CAPEC-98;
classtype:Downloader; sid:100000004; rev:1;)
```

detects the Gandcrab downloading through the attached file (i.e., Action 2). This signature corresponds to the ASTD *Downld*. It targets the inbound HTTP traffic

4. https://depot.gril.usherbrooke.ca/fram1801/iASTD-public

(*$HOME_NET*) directed to the C&C server (*$EXTERNAL_NET*). The clause *any* means that the signature accepts all connection ports from the inbound traffic. Within packet flows, it checks if the HTTP method is *GET*, the HTTP uri contains pattern *.exe HTTP/1.*, the HTTP connection is kept alive, the Accept header is used, the user-agent is Mozilla and the Host header contains the ip address of the C&C server. The unique pattern *.exe HTTP/1.* precisely characterizes the downloader trojan.

3.3.3 Zeek specification

We have specified the Gandcrab phases using Zeek scripts and signatures [5]. Zeek signatures [246] do essentially pattern matching like Snort (see Appendix A.3). The strength of Zeek is shown using Zeek scripts. The script below is a low-level representation of ASTD automatons *Downld* (CAPEC-98) and *Net_Exec_Code* (CAPEC-549).

5. https://depot.gril.usherbrooke.ca/fram1801/iASTD-public

```
...
#BLOCK 0
export {
  ...
  global v1: bool = F;
  global v2: bool = F;
  global state_downld: int = 0;
  global state_net_exec_code: int = 2;
  ...
}
...
event http_request (c: connection, ...)
{
 #BLOCK 1 (CAPEC-98)
 local g0: bool = (c$http$method == "GET")}
  && (/\/[a-z]+\.(bin|exe)/ in c$http$uri)
  && (/Mozilla/ in c$http$user_agent)
  && (/([0-9]{1,3}\.){3}[0-9]{1,3}/
   in c$http$host);

 if (state_downld == 0 && g0)
 {
   A0(v1, v2);
   state_downld = 1;
 }
 #BLOCK 3 (CAPEC-549)
 local g2: bool = (c$http$method=="GET")
  && (|c$http$uri| == 1)
  && (/[a-z]+\.(bit|ru)/ in c$http$host);} # end http_request

if (state_net_exec_code == 3 && g2)
{
  state_net_exec_code = 4;
}
#BLOCK 4 (CAPEC-549)
local g3: bool=(c$http$method=="POST")
 && (/[a-z]+\.(bit|ru)/ in c$http$host)
 && (/Mozilla/ in c$http$user_agent);

if (state_net_exec_code == 4 && g3)
{

 # Issue an alert
 A1(v1, v2);
 state_net_exec_code = 5;

event dns_request (c: connection, ...)
{
 #BLOCK 2 (CAPEC-549)
 local g1: bool =(/\x00\x01\x00\x00/
  in c$dns$query)
  && (/whatismyipaddress/
  in c$dns$query);
 if (state_net_exec_code == 2 && g1)
 {
   state_net_exec_code = 3;
 }
} # end dns_request
```

Similar to ASTD *Gandcrab_Ransom*, the header of the script contains two global variables *v1*, *v2* to notify that the downloading phase is done (see block 0). The header also declares two state variables *state_downld* and *state_net_exec_code* for maintaining states during the correlation. In the body of the script, we have two Zeek event functions: **event** *http_request(c: connection, ...)* and **event** *dns_request (c: connection, ...)*. They respectively catch HTTP and DNS requests. Within the *http_request* event, the block 1 detects the Gandcrab's action 2. The action function A0 is executed to set variables *v1* and *v2* to true. Within the *dns_request* event, the block 2 targets action 3. Within the *http_request* event, the block 3 detects action 5. Next, the block 4 checks the malicious action 6.

3.4 Execution of attack specifications

The execution process of Snort signatures is described in [271], and Zeek scripts in [246]. In Fig. 3.4, the ASTD-based detection process is shown. In a corporate network, cyber-analysts specify attacks using the graphical editor *e*ASTD and following the attack strategy methodology provided in 3.2. They also create new custom event types (e.g., sFlow, DNP3 Session) using the ontology editor Protégé. These event types are parsed into JSON to feed *e*ASTD and *i*ASTD[6]. ASTD attack specifications are saved in the local host in a specific repository, depending of the attack domain (network, host, both). A watcher agent automatically synchonizes local specifications and new event types to a remote network node, where *i*ASTD is installed. *i*ASTD has five modules.

The *capture* module collects different event sources in offline and online mode. In offline mode, it reads pcap files (option *-pcap*) and log files (option *-i*). Only log files containing Windows/Syslog traces in the ontology format are recognized. The attack specfication can be run from command line (e.g., *./iASTD -s my.spec -pcap my.pcap*), or in fully automated mode. In online mode, the *capture* module collects network streams (HTTP sessions, DNS sessions, or custom sessions) from network interfaces provided in a YAML configuration file. It also gathers Windows/Syslog event streams on endpoints using the shipping agent *hcap*. The agent *hcap* is installed on multiple endpoints, where it scans log files and sends real-time events to *i*ASTD on port 9092. The *capture* module also collects network flows/sessions from *astd_producer*, a shipping agent based on the rdkafka library. The agent *astd_consumer* allows one to consume Kafka events to feed *i*ASTD.

The *decoding/encoding* module essentially identifies which type of event stream is being read (decoding) and structures it in the corresponding ontology format (encoding). The *static analysis* module is based on the Flex/Bison analyzer and it checks if the input structured events and attack specifications are syntactically and semantically corrects i.e., well parenthesized, structured and not containing unknown keywords. Next, this module extracts the hierachichal structure of ASTD specifica-

6. The tools are available at https://depot.gril.usherbrooke.ca/fram1801/iASTD-public

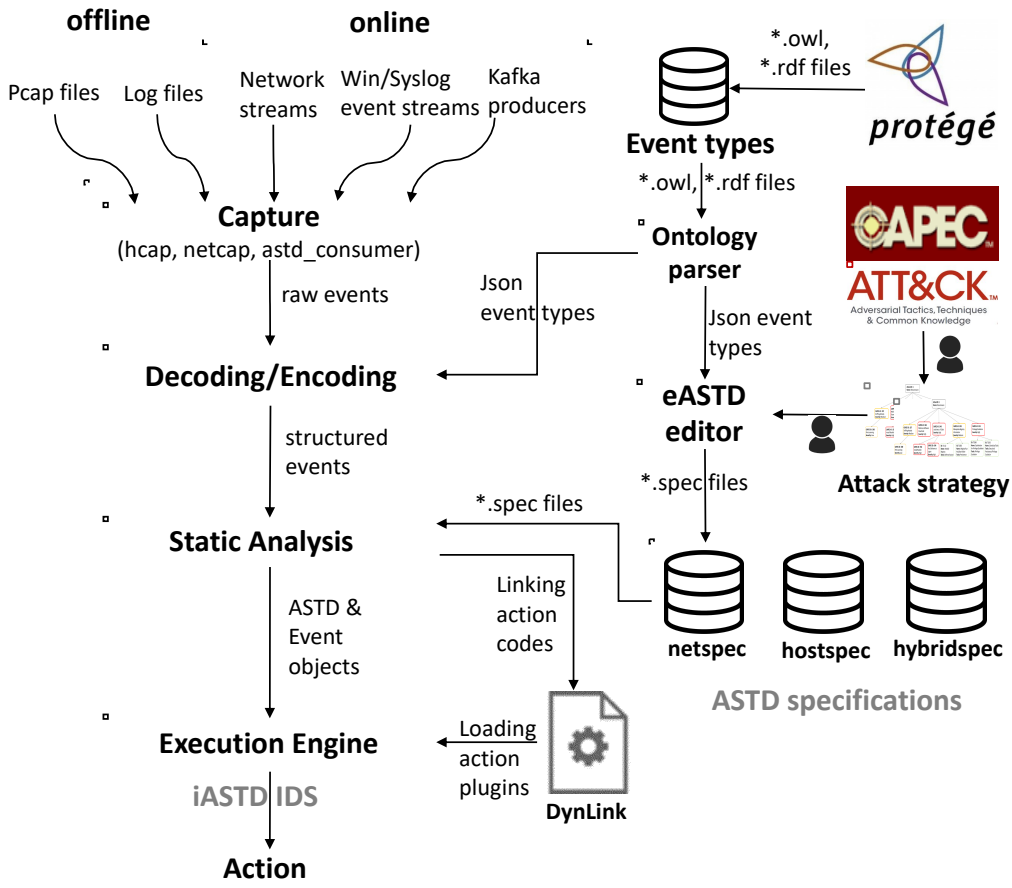


Figure 3.4 – ASTD-based detection process

tions and stores into ASTD objects. It also stores event contents into event objects. Since transition actions contain executable code, they are compiled and linked at runtime using the DynLink library. Next, they are loaded as plugin modules in the execution engine at runtime for intrusion detection.

The *execution engine* runs ASTD objects on events using semantic rules. The semantic rules for ASTDs are provided in [234]. The *i*ASTD detector can efficiently execute ASTD specifications on event streams in $n \log m$, where n is the size of the ASTD specification and m the size of the quantification variable types [95]. Once an attack behavior is detected, it executes real-time actions such as alerting, blocking of a port, or dropping of a malicious traffic.

3.5 Experiments

We have selected two existing real-world datasets (i.e., CSE-CIC-IDS2018 [293] and CTU [115]) and we have built a testbed close to a real world environment for evaluation. CSE-CIC-IDS2018 [293] is a huge dataset of terabytes of packet captures and audit traces (normal, attack), where 18 attacks were executed (including GoldenEye, HOIC DDoS HTTP, LOIC DDoS UDP, SQL Injection, XSS, FTP and SSH BruteForcing) on the Amazon Web Service (AWS) platform. CTU [115] is a dataset of gigabytes of botnet traffic (normal, background), where 7 botnets have been executed (Neris, Virut, Donbot, Sogou, Qvod, Rbot, NSIS.ay). For the real-time testbed, 20 attacks (including Gandcrab, TeslaCrypt, WannaCry and Petya) were specified in the IDS tools and executed on the AWS platform.

3.5.1 Traffic and audit data generation

For the real-time testbed, we have selected 14 services to simulate random normal user and attacker behaviors including HTTP, HTTPS, SSH, SMTP, TELNET, and FTP. Normal users perform benign activities including accessing HTTP/HTTPS pages and sending/consulting emails. Concurrently, we semi-randomly run each attack in different time frames to ensure that it is close to real-world attacks. This means that attacker can repeat the same phase or the previous one in another phase in different time intervals.

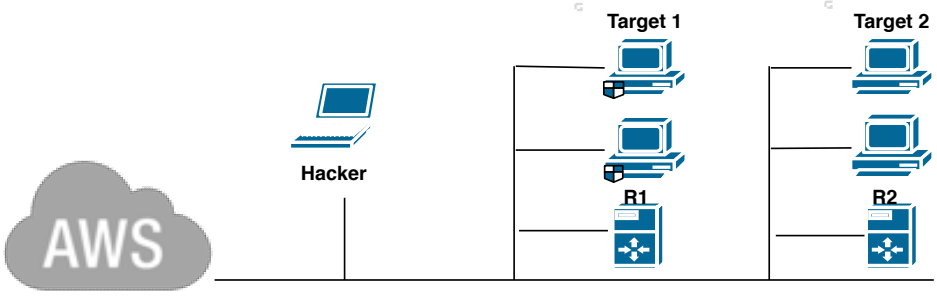


Figure 3.5 – AWS Testbed

The simulation network consisted of 2 work groups, each connected through 2 router servers (i.e., *R1* and *R2*), running on Ubuntu 16.04 (see Fig. 3.5). Each

work group had 2 local machines running on Windows 10. The first work group (containing *Target 1*) had been patched with the latest Windows updates while the second (containing *Target 2*) was running without Windows patches. The system monitor (Sysmon) has been installed on *Target 1* and *Target 2* (see Appendix A.4). Snort (version 2.9.15) and Zeek (version 3.0.0) were installed on *R1* and controlled the inbound traffic directed to the first work group. Kafka and *I*ASTD were installed on *R2* to collect and analyze Windows/Syslog events from work groups and router servers.

The network infrastructure was built on Amazon Elastic Compute Cloud (Amazon EC2) using T2 Small and Medium instances. The built-in network had a bandwidth of 550 Mbits/s while all the aforementioned services were running.

3.5.2 Results

We consider two metrics to compare the accuracy and performance of IDS tools: detection rate (DR) and false positive rate (FPR). The detection rate (DR) is the probability that the IDS outputs an alert when there is an intrusion. The false positive rate (FPR) is the probability that the IDS outputs an alert although the behaviour of the system is normal. These metrics are expressed of the form,

$$\mathsf{DR} = \frac{\mathsf{TP}}{\mathsf{TP} + \mathsf{FN}} \qquad\qquad \mathsf{FPR} = \frac{\mathsf{FP}}{\mathsf{FP} + \mathsf{TN}}$$

where False Positive (FP) is the number of normal alerts misclassified as malicious, True Positive (TP) is the number of malicious alerts correctly classified as malicious, False Negative (FN) is the number of malicious alerts misclassified as normal, and True Negative (TN) is the number of normal alerts correctly classified as normal.

Existing datasets. Results for CSE-CIC-IDS2018/CTU datasets and the real-time testbed are reported in Table 3.1. The notation *Zeek-sig/Zeek-script* is used to distinguish results of two specification versions for Zeek. Zeek-sig denotes a specification that uses only signatures while Zeek-script denotes a specification using scripts.

Table 3.1 – Evaluation of IDS tools

CSE-CIC	Zeek-sig/Zeek-script						Snort						*i*ASTD					
-IDS2018	TP	TN	FP	FN	**DR(%)**	**FPR(%)**	TP	TN	FP	FN	**DR(%)**	**FPR(%)**	TP	TN	FP	FN	**DR(%)**	**FPR(%)**
LOIC DDoS UDP	0/1	0/0	164/0	0/0	0/100	100/0	8904	4914	912	0	100	15.65	1	0	0	0	100	0
HOIC DDoS HTTP	0/1	0/0	289342/0	0/0	0/100	100/0	197	5071	755	0	100	13.31	1	0	0	0	100	0
SSH BruteForce	0/1	0/0	94216/0	0/0	0/100	100/0	67	94	0	0	100	0.00	1	0	0	0	100	0
FTP BruteForce	0/1	0/0	193392/0	0/0	0/100	100/0	3868	94	0	0	100	0.00	1	0	0	0	100	0
GoldenEye DoS	0/1	0/0	27751/0	0/0	0/100	100/0	1	96	8	0	100	7.84	1	0	0	0	100	0
Web BruteForce	71/1	0/0	0/0	0/0	100/100	0/0	3	2682	73	0	100	2.64	1	0	0	0	100	0
XSS BruteForce	19/1	0/0	0/0	0/0	100/100	0/0	1	2482	0	0	100	0.00	1	0	0	0	100	0
SQL Injection	15/1	0/0	0/0	0/0	100/100	0/0	2	2482	0	0	100	0.00	1	0	0	0	100	0
CTU																		
Neris	3/1	0/0	0/0	0/0	100/100	0/0	1	131	2	0	100	1.50	1	0	0	0	100	0
Rbot	2/1	0/0	0/0	0/0	100/100	0/0	10	5249	77	0	100	1.45	1	0	0	0	100	0
Rbot DoS	4/1	0/0	0/0	0/0	100/100	0/0	3	6684	20	0	100	0.30	1	0	0	0	100	0
Virut	2/1	0/0	0/0	0/0	100/100	0/0	1	11	0	0	100	0	1	0	0	0	100	0
Donbot	178/1	0/0	0/0	0/0	100/100	0/0	91	92	10	0	100	9.8	1	0	0	0	100	0
Sogou	2/1	0/0	0/0	0/0	100/100	0/0	1	15	0	0	100	0	1	0	0	0	100	0
qvod	2/1	0/0	0/0	0/0	100/100	0/0	2	501	36	0	100	6.7	1	0	0	0	100	0
NSIS.ay	3/1	0/0	0/0	0/0	100/100	0/0	3	97	0	0	100	0.00	1	0	0	0	100	0
Real-time																		
WannaCry	6/3	0/0	0/0	0/0	100/100	0/0	6	2434	58	0	100	2.32	2	0	0	0	100	0
Petya	13/4	0/0	0/0	0/0	100/100	0/0	22	1336	47	0	100	3.40	2	0	0	0	100	0
TeslaCrypt	4/1	0/0	0/0	0/0	100/100	0/0	8	20	0	0	100	0	2	0	0	0	100	0
Gandcrab	4/1	0/0	0/0	0/0	100/100	0/0	10	39	0	0	100	0	2	0	0	0	100	0
Normal	0/0	0/0	0/0	0/0	0/0	0/0	0	0	0	0	0	0	0	0	0	0	0	0

Overall, Zeek-script/Snort/*i*ASTD sucessfully detected CSE-CIC-IDS2018 and CTU attacks with a DR of 100%. Zeek-script produced less TPs than Zeek-sig per attack thanks to its correlation capabilities using global state variables. In Zeek-sig, we have attempted to specify signatures for Distributed DoS (DDoS) and SSH/FTP Brute attacks, essentially based on protocols and weak observed contents (e.g., `GET / HTTP/1`, `User-Agent: Mozilla`) that were not unique enough (FPR = 100%). In addition, Zeek-sig detected SQL injection, XSS and Web BruteForce with a DR of 100%, because more precise and unique patterns were found (e.g., `.php?id=3+`, `script\x25 \x33\x45`).

Snort generated a significant FPR for LOIC DDoS UDP (15.65%) and HOIC DDoS HTTP(13.31%). Since these attacks have not unique signatures (e.g., `GET /`) , one must rely on the statistical distribution of packets per second. We have used Snort features like *threshold* to reduce the number of alerts. Snort also generated a significant number of TPs for LOIC DDoS UDP (8904), HOIC DDoS HTTP (197), FTP BruteForce (3868), SSH BruteForce (67), and Donbot (91).

Overall, Zeek-script and *i*ASTD achieved better detection performance than Zeek-sig and Snort with a high DR of 100% and no FP. The tools were able to correlate multiple HTTP and DNS connections from CSE-CIC-IDS2018 and CTU attacks.

Real-time testbed. In Table 3.1, Zeek/Snort/*i*ASTD detected ransomware attacks with a DR of 100%. Zeek-script produced less TPs than Zeek-sig and no FP for WannaCry and Petya attacks. For Zeek-script, we got 3 TPs for WannaCry and 4 TPs for Petya using a behavioral analysis over SMB based on the entropy [7]. In addition, Zeek-script did not generated FPs on the normal traffic.

Like Zeek-sig, Snort matched network sessions only on a stream-by-stream basis and generated redundant TPs per attack (e.g., 22 TPs for Petya). In addition, Snort produced a significant FPR compared to Zeek-script and *i*ASTD for WannaCry and Petya attacks (2.32% for WannaCry, 3.4% for Petya).

Oppositely to Snort and Zeek, *i*ASTD can correlate both network sessions and host logs by generating no FP and 1 TP per attack for each environment (i.e., 1 TP for host and 1 TP for network). Similar to Zeek and Snort, *i*ASTD did not generate FPs on the normal traffic.

3.5.3 Discussion

Snort is a low level, stateless, event pattern language. Zeek is a scripting language that is essentially a programming language. Its composition mechanisms are those of imperative programming languages: procedural abstraction and programming composition using if-then-else, case analysis, and state variables. Creating Zeek script is a daunting, complex and error-prone task. ASTD is a more abstract language. Its state machines offer a graphical, deep representations of attack behavior and state transitions. Its process algebra operators free the specifier from dealing with low level composition of attack specification elements. An attack can be specified in a modular fashion, following the natural language description of an attack's structure into phases and steps. Attack specifications can be easily composed to create a global IDS specification. Snort and Zeek signatures can be easily represented by ASTD specifications using automatons and quantified interleaves.

Verification tools [234] can be used to check the correctness of ASTD specifications and check properties about them. ASTDs are also extensible, portable and heterogeneous as they are domain-independent. This means that they can be executed in any

7. https://depot.gril.usherbrooke.ca/fram1801/iASTD-public

environment and on any source (e.g., network/host events, natural events). It is an important factor to deal with complex attacks that operate on common networks but also on cyber-physical systems. Snort and Zeek languages require additional updates of the source code to be extended in new environments (e.g. host). In addition, Zeek and Snort signatures refer to unique strings (e.g., `GET /wordpress/?ARX8`) that can easily be changed by the attacker and make them obsolete. ASTD operators like quantified interleaving allow one to abstract from a particular machine (ip address, name) and to specify any ordering constraint, at any level of abstraction (e.g., ip, host name, uuid, etc.) and in any combination.

Zeek is stateful using state variables and event functions. It can correlate multiple network events and latterly host events after some manual updates of the source code. Being abstract and domain-independent, the stateful language ASTD correlates multiple diverse events in any environment, thanks to process algebra operators and ontologies that are used on transitions of ASTD automata to structure the knowledge about events. Snort cannot correlate different network connections (e.g., HTTP, DNS, SSH). Snort features like *flowbits* can only handle packets within the same connection.

As measured in our experiments, Snort has a significant FPR and a high DR on average. The processing time of Snort on the real-time testbed is also very low on average (2.336s for 1Gb packets, 8.201s for 10Gb packets). Zeek-script has a very low FPR and a high DR on average. Its processing time on the real-time testbed is low on average (7.480s for 1Gb packets, 29.766s for 10Gb packets). *I*ASTD has a very low FPR and a high DR average but its processing time on the real-time testbed is relatively medium on average (20.184s for 1Gb packets, 96.778s for 10Gb packets). For network intrusion detection, Snort is faster than Zeek and *I*ASTD since it matches each single network connection without correlating them. Zeek can correlate multiple network connections and it is faster than *I*ASTD. For network and host intrusion detection, *I*ASTD was able to correlate both network connections and Windows/Syslog events but the huge amount of events affected considerably the detection time (188.667s for 10Gb packets and 87 450 mixed Windows/Syslog events).

To improve the processing time of *I*ASTD for network detection, we are currently working on translation rules to generate Zeek scripts from ASTD specifications [8].

8. The translation rules and the compiler are available at https://depot.gril.

Hence, one could use the Zeek engine to run ASTD specifications on network event streams. The Zeek scripts generated from ASTD specifications are as efficient as manually written Zeek scripts. Another way is to generate Snort signatures from ASTD specifications to process network events faster. This approach involves several FPs due to the aforementioned limitations of Snort. Thus, it is not suitable for network detection.

3.6 Conclusion

We have compared Zeek, Snort, and ASTD for intrusion detection. Snort is a stateless language that offers very limited event correlation capabilities. Both Zeek scripts and ASTD are stateful and thus better support event correlation. Consequently, Snort produces more redundant true positives and false positives than Zeek and ASTD. Zeek scripts and ASTD are equivalent in terms of detection and correlation capabilities. However, Zeek being a scripting language, it is less abstract than ASTD. Thanks to its process algebra composition operators, ASTD makes it easier to create, reuse, compose and maintain attack specifications. Snort is the most efficient IDS in terms of processing time because it does not support correlation. Zeek is faster than *i*ASTD, but it should be possible to compile ASTD specifications into Zeek scripts to execute ASTD specifications more efficiently while benefitting from the features of the Zeek execution environment.

usherbrooke.ca/lionel-tidjon/castd

Chapitre 4

Compilation des ASTDs

Résumé

Le langage de spécification ASTD (Algebraic State-Transition Diagram) est une notation graphique, exécutable, modulaire et indépendante du domaine qui permet la composition de machines à états hiérarchiques à l'aide d'opérateurs d'algèbres de processus. Les spécifications ASTD peuvent être exécutées sur un interpréteur appelé iASTD. Cependant, les performances de l'iASTD sont considérablement affectées lors du traitement de spécifications ASTD de grandes tailles sur des flux d'événements volumineux. Dans cet article, nous proposons une méthodologie qui génère du code exécutable à partir des spécifications ASTD en utilisant des règles de traduction, via un langage intermédiaire. Ces règles sont implémentées dans un outil appelé cASTD qui génère des programmes en C++ ; des programmes dans d'autres langages similaires comme Java pourraient également être générés à partir du langage intermédiaire. L'exécution est plus rapide que iASTD et d'autres outils comme Beepbeep v3 [320], Larva [68], et MonPoly [35].

Commentaires

La contribution ici réside dans la conception, l'implémentation, l'optimisation, et l'évaluation d'un compilateur du langage ASTD vers les langages de programmation tels que C++ ou Java. Une syntaxe du langage intermédiaire a été définie ainsi que les règles traduction des ASTDs vers le langage intermédiaire et du langage intermédiaire vers un langage de programmation donnée. Le code généré a été optimisé avec l'élimination des calculs redondants, des branchements inutiles et le support de la Kappa optimisation pour l'exécution efficiente des opérateurs quantifiés. Les programmes compilés ont été comparés avec plusieurs outils de traitement de flux d'événements (i.e., iASTD, Beepbeep v3, Larva, MonPoly), sur des ensembles de données de référence.

Les contributions décrites dans ce chapitre ont fait l'objet d'un journal soumis dans la revue *ACM Transactions on Programming Languages and Systems*.

Les contributions et l'article sus-cité ont été élaborées par mes soins en tenant compte des remarques et commentaires issus de mon équipe d'encadrement.

Translating ASTDs into High-Level Programming Languages

Lionel N. Tidjon
Université de Sherbrooke, GRIF, Québec, Canada
Télécom SudParis, SAMOVAR, Institut Polytechnique Paris, France
lionel.nganyewou.tidjon@usherbrooke.ca

Marc Frappier
Université de Sherbrooke, GRIF, Québec, Canada
marc.frappier@usherbrooke.ca

Amel Mammar
Télécom SudParis, SAMOVAR, Institut Polytechnique Paris, France
amel.mammar@telecom-sudparis.eu

Keywords: ASTD; Compilation; Intermediate language; Translation rules; Code generation; Event Processing; Information Security Systems; Industrial and Control Systems; Information Systems

Abstract

Algebraic State-Transition Diagram (ASTD) is a domain-independent, graphical, executable, and modular notation that allows for the composition of hierarchical state machines using process algebra operators. ASTD specifications can be executed on an interpreter called iASTD. However, the performance of iASTD is significantly affected when processing large ASTD specifications on high throughput event streams. In this paper, we propose a methodology that generates executable code from ASTD specifications using translation rules, through an intermediate language. These rules are implemented in a tool called cASTD that generates programs in C++; programs in other similar languages like Java could also be generated from the intermediate language. The execution is faster than iASTD and other runtime tools like Beepbeep v3, Larva, MonPoly.

4.1 Introduction

An Information System (IS) is composed of a collection of processes. A process run consists of a sequence of system states or events with a priori total ordering [186]. This sequence of events is also called a trace. A run of a system can be considered as an infinite trace [201] while an execution of a system is a finite trace i.e. a finite frame of a run. Runtime monitoring is the process of checking a property on a trace [30]. A property defines a language over traces. A decision for the property is made after executing traces by a device called monitor. An online monitoring consists in executing traces efficiently and in a continuous manner. The process of executing a property on finite or recorded traces is called offline monitoring.

Algebraic State-Transition Diagram (ASTD) is a modular, domain-independent, executable, and graphical notation that was introduced to extend statecharts (i.e. hierarchical state machines) with process algebra operators such as choice, sequence, parameterized synchronization, and quantified interleaving [98, 235]. It allows one to capture an holistic view of activities on the monitored system and to define complex relationships between events (e.g. event correlation). In addition, ASTD allows for formal proof and verification of specifications by translating them into B/Event-B as described in [100, 217]. In [235, 313], ASTDs have been used to check cyber-attack behaviours over offline and online event traces.

ASTDs are supported by a graphical editor called eASTD and an interpreter called iASTD for execution [313]. The performance of iASTD is considerably affected when processing large ASTD specifications on big heterogeneous event streams [313]. In this paper, we propose a methodology to compile ASTD specifications into an Intermediate Language (IL) that can be translated into high-level programming languages, like C++ or Java, and optimized to improve execution efficiency. The translated code is later compiled into monitoring programs using native compilers (e.g., GCC, JIT). The methodology is implemented in a tool called cASTD (see Fig. 4.1).

We have compared the execution time of ASTD specifications with cASTD and iASTD on large specifications and large event streams. We have also compared cASTD and iASTD with other runtime verification tools (Beepbeep v3, MonPoly). cASTD provides a 10x performance improvement over iASTD and other tools.

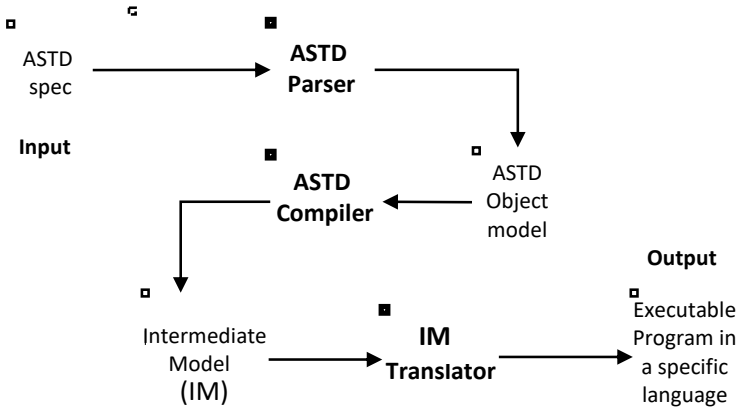


Figure 4.1 – Compilation methodology

The rest of this paper is structured as follows. Section 4.2 presents an introduction to ASTDs. The methodology on the compilation of ASTD specifications conventional imperative languages is described in Section 4.3. Section 4.4 shows an implementation of the methodology through the tool cASTD. In Section 4.5, we present existing runtime monitoring work and tools that will be compared to cASTD in terms of performance. The evaluation of generated code efficiency for cASTD and existing tools is described in Section 4.6. In Section 4.7, we conclude with some perspectives.

4.2 Introduction to ASTDs

The ASTD notation [235, 310] allows for the composition of automata using operators sequence, choice, Kleene closure, guard, parameterized synchronization, flow (the AND states of Statecharts) and quantified versions of parameterized synchronization and choice. Each ASTD operator defines an ASTD type that can be applied to sub-ASTDs of any type. Elementary ASTDs are defined using automata. Automaton states can either be elementary or composite; a composite state can be of any ASTD type. In comparison, Statecharts only allow for composite states of type OR or AND. ASTD types share common properties which are represented in the abstract type ASTD, and from which all other ASTD types inherit. ASTD supports the declaration of attributes (i.e., state variables) and actions that can modify these attributes when a transition is executed. Attributes can be locally declared within

each ASTD. Actions are declared on automaton states, automaton transitions, and at the level of an ASTD itself for execution on all of its transitions. An ASTD can also have parameters to be called from another ASTD.

4.2.1 A Simple Example

Fig. 4.2(a) describes an ASTD A of type flow ($\Cup$). This flow operator combines two sub-ASTDs B and E as follows. When an event is received, both B and E try to execute it, independently. If they succeed, they each make an internal transition, irrespective of the capability of the other (ie, there is no synchronisation between B and E). This operator is similar to an AND state in Statecharts.

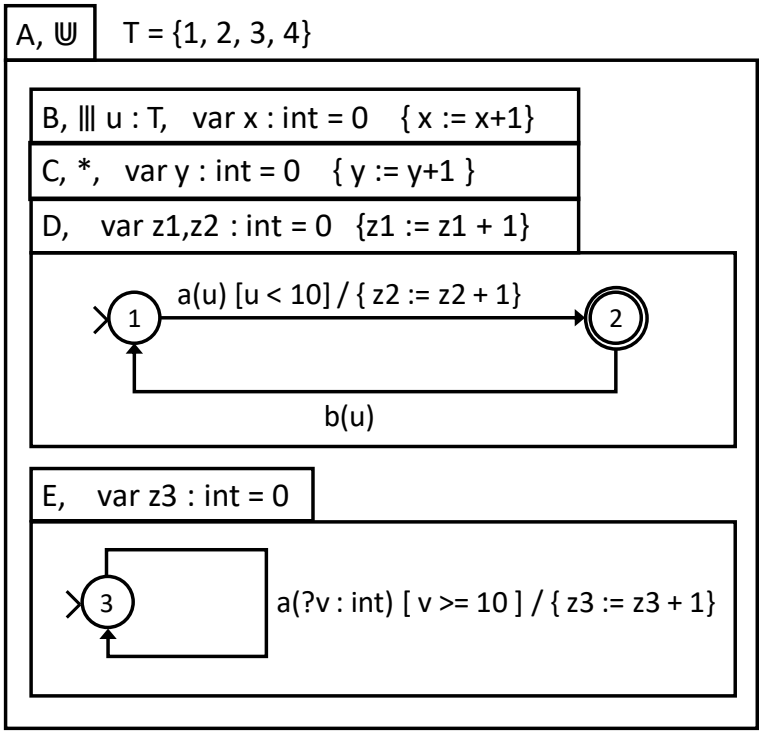
ASTD B is a quantified interleave ($|||$). It declares a read-only quantified variable u and it instantiates its sub-ASTD C for each possible value v_i of u in T, i.e.,

$$B \;=\; |||u : T : C \;=\; [u := v_1]\ C\ |||\ \ldots |||\ [u := v_n]\ C$$

These instances execute in interleaving. Basic types (integer and string) and user-defined types can be used for typing variables. ASTD B declares an attribute (i.e., a state variable) x of type integer which is initialized to 0. It also declares an action x *:= x+1*, which is executed for each transition of B. Thus, the value of x denotes the number of transitions that B has executed.

ASTD C is a Kleene closure. It iterates on its sub-ASTD D, which is an automaton. ASTD C can start a new execution of D (i.e., restart D from its initial state) whenever D is in a final state. ASTD C declares an attribute y, and an action $y := y+1$. Since y is declared within a quantified interleave, there is a copy of y for each value of u. Thus, y denotes the number of transitions for each instance of C.

The initial state of automaton D is 1. This automaton declares two attributes *z1* and *z2*. A transition of an automaton has the following form: $\sigma(p_1, \ldots, p_n)\ [g]\ /\ act$, where σ is an event label, each p_i is either a variable, a constant or the wild-card "_", which accepts any value. When an event $e(v_1, \ldots, v_n)$ is received by D, the current state of D is checked to see if a transition labelled with e exists and whether $p_1, \ldots, p_n$ verify the equality $(v_1, \ldots, v_n) = [\Gamma](p_1, \ldots, p_n)$ an execution environment which contains the current value of enclosing variables (i.e., ASTD parameters, at-



(a)

			u=1			u=2			
	Event	x	y	*z1*	*z2*	y	*z1*	*z2*	*z3*
0	Init.	0	0	0	0	0	0	0	0
1	*a(1)*	1	1	1	1				
2	*b(2)*								
3	*b(1)*	2	2	2					
4	*a(1)*	3	3	3	2				
5	*a(2)*	4				1	1	1	
6	*a(1)*	5	4	1	1				
7	*b(1)*	6	5	2					
8	*b(2)*	7				2	2		
9	*a(10)*								1

(b)

Figure 4.2 – Example of ASTD specification and execution trace

tributes and quantification variables). If so, the guard g of the transition is checked; if it is satisfied, the transition is triggered and the transition action *act* is executed. If no transition can be triggered in each automaton of the ASTD, then the event is ignored, and the next event in the input stream is processed. If several transitions can be triggered, one is non-deterministically chosen.

The transition from State 1 to State 2 is labelled by event *a(u)*. Recall that u is a quantified variable, thus it has been instantiated with a concrete value in each instance of C. The transition includes a guard $u < 10$ and an action $z2 := z2 + 1$. State 2 is a final state. Attribute *z1* denotes the number of transitions in the current iteration of the Kleene closure C, whereas y denotes the total number of transitions for C. The guard $u < 10$ entails that all events *a(v)* and *b(v)* such that $v \geq 10$ will be ignored by B. Attribute *z2* denotes the number of transitions on a in the current iteration of the Kleene closure.

ASTD E is also an automaton with attribute *z3*. Its only transition is labeled by event *a(?v : int)*, which declares a local variable v of type *int*. Such a declaration matches any value for v; the scope of v is the transition itself (ie, guard and action, in read-only). It also includes a guard $v \geq 10$, such that E only processes events that B cannot process. Attribute *z3* counts the number of events *a(v)* such that $v \geq 10$.

Fig. 4.2(b) describes an example of execution of ASTD A with the value of ASTD attributes. Only the active instances of C are illustrated, the others being at their initial value. An empty cell means no change to the value. First, the ASTD states and attributes are initialized on Line 0. ASTD A accepts the first input *a(1)* on Line 1; it is executed on instance $u = 1$ of C. The second input *b(2)* is rejected, because it can only be accepted by instance $u = 2$ of C, but this instance is still in its initial state, since *a(2)* has not been received yet. All the subsequent inputs are accepted. On Line 6, note that a new iteration has started for instance $u = 1$ of C. Indeed, the automaton D is in State 2, after accepting *a(1)* on Line 4. In State 2, D cannot accept *a(1)*, but since State 2 is final, the Kleene closure of C can start a new iteration in State 1 and accept *a(1)*; all attributes of D are re-initialized when starting a new iteration.

4.2.2 ASTD Abstract Syntax

The abstract syntax of the ASTD is defined in [197, 235]. We provide here a subset of this syntax, for illustration purposes. As stated earlier, each ASTD type inherits from an abstract type ASTD which introduces properties shared by all ASTD type (see Sect. 2.2). Automatons are elementary ASTDs. An ASTD can be unary or binary. Unary ASTDs consist of Kleene, guard, quantified choice, and quantified synchronization [310]. Binary ASTDs consist of choice, flow and parameterized synchronization [310]. The abstract representation of an ASTD has the following structure $\mathsf{ASTD} \triangleq \langle name, P, Attr, Act_{astd} \rangle$, where $name$ is the name of the ASTD, P is an optional list of parameters, $Attr$ is a list of attributes, Act_{astd} is an action. Parameters P are used to receive values passed by a calling ASTD; they can be read-only or read-write. Attributes $Attr$ are state variables that can be modified by actions within the scope of the ASTD and read in guards. $Attr$ is a list of attributes v of the form $\langle name, T, init \rangle$, where T is the type of v and $init$ its initial value. Quantified variables are of the form $\langle name, T \rangle$ and they read-only; they can not be modified by actions.

4.3 Methodology

In this section, we describe the compilation methodology of ASTD specifications into several programming languages. It is divided into 4 steps: parsing, translation from the ASTD language to the Intermediate Model (IM), translation from the IM model to the target code, and code optimization (see Fig. 4.1). The first component reads an ASTD specification in JSON and produces an ASTD object model. The second one generates a program represented into an intermediate language. The last one translates an intermediate model of the program into a concrete programming language like C++ or Java. Several optimizations such as removing redundant calculations are applied to the output code during generation of the program.

4.3.1 Translation of ASTDs into an Intermediate Model

This step consists in encoding an ASTD meta-model into a model written into an intermediate language (IL). We first define the syntax of the IL language. Next, we describe formal translation rules to produce the IL model conforming to the meta-model.

4.3.1.1 The Intermediate Language

The intermediate language (IL) is a simple programming language which can be translated into conventional programming languages like C++ or Java. A model written into the IL language has a name, a set of variable declarations, and a set of functions.

Syntax. The grammar of the IL language is provided in Fig. 4.3, with the usual grammatical operators "|" (choice), "⟦ ⟧" (optional), "+" (at least one) and "*" (zero or some), with comma separated occurrences for "+" and "*". IL language keywords are in **roman bold**, and lexical elements like identifiers are in sans serif.

The IL model consists of three sets of declarations: types, variables, and functions. The type declarations are similar to structures in C (i.e., **record**). We also use the sum operator "⊕" which denotes the sum of two types (i.e., variant records) The functions are of the form ⟨*name, params, type, block*⟩, where *name* is the identifier, *params* is an optional list of input parameters, *type* is an optional return type and *block* is the body of the function. The body is composed of a list of statements.

The conditional statement **if** $C_1 \rightarrow S_1, \ldots, C_n \rightarrow S_n$ **fi** is used instead of **if** C_1 **then** S_1 **else if** C_2 **then** $S2 \ldots$ **end** to simplify the representation of complex ASTD cases. The **if-fi** statement is Dijkstra's selection statement; we have added the optional case **else** $\rightarrow$ *Statement*, which is an abbreviation for **not** (C_1 **or** C_2 ... C_n). An **if-fi** statement aborts when the else is absent and none of the conditions holds. The call ⟨*name, callparams*⟩ allows to invoke the function *name* with optional parameters *callparams*. Test operator like **or**, **and**, **not**, **eq** and **neq** denote usual Boolean operators. Other operators like exists are used to check whether there exists a value for a variable from a given set that satisfies a given condition, and forall, that checks whether a condition is satisfied for all values of a set.

$$
\begin{aligned}
ILModel &::= \text{identifier } Body \\
Body &::= \textbf{begin } Decl^* \ Function^* \ \textbf{end} \\
Decl &::= TypeDecl \mid VarDecl \\
TypeDecl &::= \textbf{def type } \text{identifier } = \ \textbf{record \{ } VarType^+ \ \textbf{\}} \\
Function &::= \text{identifier } (VarType^*) \ Type \ FuncBlock \\
&\mid \text{identifier } (VarType^*) \ FuncBlock \\
FuncBlock &::= \textbf{begin } Statement^* \ \textbf{end} \\
Statement &::= VarDecl \\
&\mid \textbf{if } BlockIf \ \textbf{fi} \\
&\mid \textbf{while } Condition \ \textbf{do } Statement \ \textbf{endwhile} \\
&\mid \textbf{for } Condition \ \textbf{do } Statement \ \textbf{endfor} \\
&\mid Identifier \ := \ Expression \\
&\mid Call \\
VarDecl &::= \textbf{var } VarDecl' \mid \textbf{const } Vardecl' \\
VarDecl' &::= VarType \mid VarType \ = \ Value \\
VarType &::= \text{identifier } : \ Type \\
Type &::= \text{identifier} \mid \text{identifier } \oplus \text{ identifier} \\
Value &::= \text{number} \mid \text{string} \mid \text{jsonObject} \\
BlockIf &::= Condition \ \rightarrow \ Statement \\
&\mid Condition \ \rightarrow \ Statement \ [] \ BlockIf \\
&\mid Condition \ \rightarrow \ Statement \ [] \ \textbf{else} \ \rightarrow \ Statement \\
Condition &::= PredCond \mid Boolean \\
PredCond &::= (\textbf{and } Condition^+) \\
&\mid (\textbf{or } Condition^+) \\
&\mid (\textbf{not } Condition) \\
&\mid (TestOperator \ Expression^+) \\
Expression &::= (NumOperator \ Expression^+) \\
&\mid Call \mid Number \mid Boolean \mid \text{identifier} \\
Call &::= \text{identifier } (\text{identifier}^*) \\
NumOperator &::= + \mid \ - \ \mid \backslash \mid * \\
TestOperator &::= \textbf{eq} \mid \textbf{neq} \mid \textbf{leq} \mid \textbf{geq} \mid \textbf{lt} \mid \textbf{gt} \mid \textbf{in} \mid \text{exists} \mid \text{forall} \\
Boolean &::= \text{true} \mid \text{false}
\end{aligned}
$$

Figure 4.3 – The grammar of the intermediate language

4.3.1.2 The Structure of the Generated Programs

The translation algorithm is based on the following four translation functions:

— $\tau_{type}(A)$ generates the type and constant declarations needed to represent the state of ASTD A;

— $\iota(A)$ generate the initialization statements for the state of ASTD A;

— $\phi(A)$ generates a condition that determines if the current state of A is final;

— $\tau_{event}(\sigma, A)$ generates the body of a function that determines if an event of label σ can be executed by A, and updates the state of the ASTD if so.

The generated program contains a single global variable, called ts_A of type TState_A. The type of this variable is generated by function $\tau_{type}(A)$. Function *main* of the generated program has the following form.

$$main(P) \triangleq \tag{4.1}$$

begin

$\iota(A)$; (4.2)

while true **do**

e := read_event(src);

if (4.3)

[[ ] (**eq** $e.label$ σ) $\rightarrow$ $\sigma(e.P)$]]$_{\sigma \in \Sigma_A}$ (4.4)

[] **else** msg("Event is not recognized") (4.5)

fi

endwhile

end

Symbol P on Line (4.1) denotes the parameters of *main* and it consists of the parameters of ASTD A. On Line (4.2), the state of the ASTD is initialized. Then, the program loops indefinitely to read events e from an external source and calls a

specific function $\sigma(P)$ for each event of label σ. The **if** statement on Line (4.3) simulates a case statement that calls the function matching event label σ. If there is no match for event e, then an error message "Event is not recognized" is thrown on Line (4.5). The notation $[\![\odot\ E(i)]\!]_{i \in \{v_1 \ldots, v_k\}}$ used on Line (4.4) denotes the expression $E(v_1) \odot \ldots \odot E(v_k)$, assuming some ordering on the values v_i when necessary. We use it to denote a list of statements or other declarations like **if-fi** statement. Symbol $\odot$ denotes either an operator or a separator like "[]".

Function $\sigma(P)$, that processes events of label σ, has the following structure.

$$\sigma\ (P) \triangleq \tau_{event}(\sigma, A)$$

Its body has the following form:

$$\begin{aligned}
&\tau_{event}(\sigma, A) \triangleq \\
&\qquad \textbf{if } C_1 \ \rightarrow\ Act_1 \\
&\qquad [] \ \ldots \\
&\qquad [] \ C_n \ \rightarrow\ Act_n \\
&\qquad [] \textbf{ else } \rightarrow\ \mathsf{msg}(\text{"Event not executable"}) \\
&\qquad \textbf{fi}
\end{aligned}$$

where C_i are conditions and Act_i are actions. We define $Cond(\tau_{event}(\sigma, A))$ as the disjunction of all conditions in $\tau_{event}(\sigma, A)$, i.e.,

$$Cond(\tau_{event}(\sigma, A)) = (\textbf{or } C_1 \ \ldots\ C_n)$$

In our translation rules, we often use variables (e.g., A) to denote some element of the ASTD metamodel. We sometimes use the value of A to construct an identifier in the generated program. For instance, we define an identifier TState_A in the generated program that represents the type for the state of an ASTD A. For the sake of clarity, we have to distinguish two cases: i) "A" that appears as a letter in a symbol, and ii) "A" as a variable whose value is used to construct an identifier. We use $\$A$ to denote the latter case. For instance, we write TState_$\$A$ to construct an identifier

that includes the name of ASTD A as its suffix.

4.3.1.3 Translation rules of each ASTD Type

In this section, we describe the translation rules for the most significant ASTD types.

Automaton ASTD. Let A be an automaton ASTD. Recall that A has the following structure:

$$\langle \mathsf{aut}, \Sigma, S, \zeta, \nu, \delta, SF, DF, n_0 \rangle$$

Σ is the alphabet. S is the set of state names of the automaton. Special state names H and H^* respectively denote shallow and deep history states of Statecharts. $\zeta \in S \rightarrow \langle Act_{in}, Act_{out}, Act_{stay} \rangle$ maps each state name to its actions: Act_{in} is executed when a transition enters the state; Act_{out} is executed when a transition leaves the state; Act_{stay} is executed when a transition loops on the state or is executed within the state. $\nu \in S \rightarrow \mathsf{ASTD}$ maps each state to its sub-ASTD, which can be elementary (noted elem) or composite. An automaton transition from n_1 to n_2 labelled with $\sigma[g]/Act_{tr}$ is represented in the transition relation δ as follows:

$$(\eta, \sigma, g, Act_{tr}, \mathit{final}?)$$

Symbol η denotes the arrow. There are three types of arrows: $\langle \mathsf{loc}, n_1, n_2 \rangle$ denotes a local transition from n_1 to n_2, $\langle \mathsf{tsub}, n_1, n_2, n_{2_\flat} \rangle$ denotes a transition from n_1 to sub-state $n_{2_\flat}$ of n_2 such that $n_{2\flat} \in \nu(n_2).S$, and $\langle \mathsf{fsub}, n_1, n_{1_\flat}, n_2 \rangle$ denotes a transition from sub-state $n_{1_\flat}$ of n_1 to n_2 such that $n_{1\flat} \in \nu(n_1).S$. Symbol *final*? is a Boolean. When *final*? = true, the source of the transition is decorated with a bullet (i.e, $\bullet$); it indicates that the transition can be fired only if n_1 is final. $SF \subseteq S$ is the set of shallow final states, while $DF \subseteq S$ denotes the set of deep final states, with $DF \cap SF = \varnothing$. $n_0 \in S$ is the name of the initial state.

The state of an automaton is a structure of type $\langle \mathsf{aut}_\circ, n, E, h, s \rangle$. $\mathsf{aut}_\circ$ is the constructor of the automaton state. $n \in S$ denotes the current state of the automaton. E contains the values of the automaton attributes. $h \in S \nrightarrow \mathsf{States}$ is the history function that records the last visited sub-state of a state. $s \in \mathsf{States}$ is state of the

sub-ASTD of n, when n is a composite state; $s = \mathsf{elem}$ when n is elementary.

State and Attributes. Let S_c denote the set of composite states of automaton A. The types generated for automaton A are defined as follows:

$\tau_{type}(A) \triangleq$ **def type** TState_A = **record {**
  autState : AutState,
  $[\![,\ v.name : v.T]\!]_{v\ \in\ A.Attr}$ (4.6)
  ts : TStateComposite_A if $S_c \neq \varnothing$
  hState : Map<AutState, TStateComposite_A>, if $\exists n \in S_c$ that contains an history state
};
const shallow_final_A : Set<AutState> $= A.SF$; if $A.SF \neq \varnothing$
const deep_final_A : Set<AutState> $= A.DF$; if $A.DF \neq \varnothing$
$[\![;\ \tau_{type}(A.\nu(n))]\!]_{n\ \in\ S_c}$ (4.7)

Variable autState stores the name of the current state of A; its type AutState is an enumeration of the element of $A.S$. On Line (4.6), the list of attribute declarations is generated. Variable ts holds the state of the current composite state, when applicable; its type TStateComposite_A is the sum of the types of the composite states, and it is defined as $[\![\oplus$ TState_n $]\!]_{n\ \in\ S_c}$. Variable hState is a map that represents the history states. It is generated when at least one composite state contains an history state. Its domain contains only the composite state that contains an history state. Constant shallow_final_A is the set $A.SF$ of shallow final states of A and deep_final_A is the set $A.DF$ of deep final states of A. On Line (4.7), the types of the composite states are generated.

Since the type generated for the state of ASTD A is a nesting of the types of its sub-ASTDs states, each reference to a field of a sub-ASTD state must be prefixed with the enclosing ASTD state. For instance, if ASTD A_1 has a sub-ASTD A_2 with attribute x, then a reference to x must be prefixed by ts_A1.ts.x . We define function λ for generating these prefixes. Let *parent* be a partial function that returns the

immediate parent of an ASTD. The function λ is defined as follows:

$$\lambda(A) = \begin{cases} \mathsf{ts\_}\$A, & \text{if } A \text{ is the root} \\ \lambda(parent(A))\ .\ \mathsf{cast}(\mathsf{TState\_}\$A,\ \mathsf{ts}), & \text{else if } \mathsf{ts} \text{ is of type sum in } parent(A) \\ \lambda(parent(A))\ .\ \mathsf{ts}, & \text{otherwise} \end{cases}$$

where $\mathsf{cast}(T, x)$ is a cast expression that returns the variable x with type T. In Fig. 4.2, the flow A is the root; then $\lambda(A) = \mathsf{ts\_}A$. To access properties of the quantified interleaving B, the prefix is $\lambda(B) = \lambda(A).\mathsf{ts} = \mathsf{ts\_}A.\mathsf{ts}$. The automaton D has no composite states (i.e., $S_c = \varnothing$) then no cast is applied. For example, suppose that state 1 and 2 are composite states, and we wanted to access some attribute x of state 2. The type of ts in the state of D is the sum $\mathsf{TState\_}1 \oplus \mathsf{TState\_}2$. The prefix of x would be given by $\lambda(2) = \lambda(D).\mathsf{cast}(\mathsf{TState\_}2, \mathsf{ts})$.

Let S_{ch} be the set of composite states that include an history state. For an ASTD automaton, the initialization function $\iota(A)$ is computed as follows:

$$\begin{aligned} \iota(A) \triangleq\ & \lambda(A).\mathsf{autState} := A.n_0; \\ & [\![;\ \lambda(A).v.name := v.init]\!]_{v\ \in\ A.Attr} \\ & [\![;\ \lambda(A).\mathsf{hState}(n) := i(A.\nu(n))]\!]_{n\ \in\ S_{ch}} \\ & \iota(A.\nu(A.n_0)) \qquad \text{if } n_0 \text{ is composite} \end{aligned}$$

It initializes the attributes of A, the history states with the initial state of the corresponding sub-ASTD, and the initial state of A. The final state function $\phi(A)$ is computed as follows:

$$\begin{aligned} \phi(A) \triangleq (\mathbf{or}\ \ & (\mathbf{in}\ \ \lambda(A).\mathsf{autState}\ \ \mathsf{shallow\_final\_}\$A) \\ & (\mathbf{and}\ \ (\mathbf{in}\ \ \lambda(A).\mathsf{autState}\ \ \mathsf{deep\_final\_}\$A) \\ & \phi(A.\nu(\lambda(A).\mathsf{autState}))\ )\ ) \end{aligned}$$

Function $\phi(A)$ checks whether A is in a shallow final state or a deep final state.

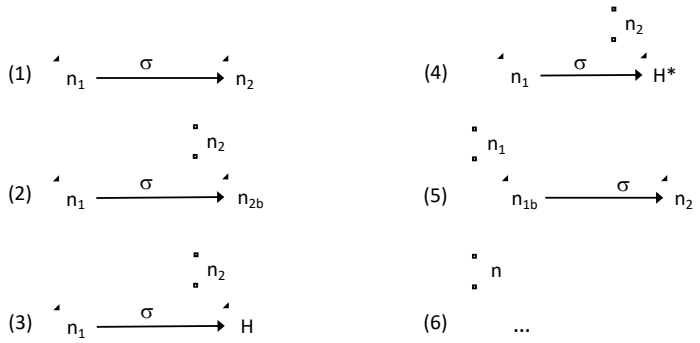


Figure 4.4 – Six cases of transitions in automata

The operation (**in** c X) checks if the element c in the set X. For a deep final state, $\phi(\,)$ is recursively called to determine if the sub-state is also final. For sake of concision and simplicity, we will omit prefix "$\lambda(A).''$ in the definitions in the sequel, as it is easy to determine when it should be used (i.e., whenever we refer to a field of a state).

Event. The translation of events for an automaton is one of the most complex cases, because the automaton notation is very rich. The semantics of automaton transitions is defined using 6 inferences rules [310] which are quite complex. We will provide the intuition of the semantics to describe and justify the translation; the reader is referred to [310] for the formal semantics of automaton behaviour. Fig. 4.4 illustrates the 6 cases of transition.

Let $T_\sigma \subseteq A.\delta$ be the set of transitions labelled with σ. Let C_t be an activation condition for the transition $t \in T_\sigma$, H_t be the statement to update the history state, Ω_t be the compositions of actions defined on automaton states and the ASTD itself, and T_t be the statements for updating the state name and its sub-ASTD. The translation of an event of label σ is computed as follows.

$$
\begin{aligned}
\tau_{event}(\sigma, A) \;\triangleq\; & \textbf{if} \\
& [\![\ C_t \;\rightarrow\; \Omega_t \;;\; T_t \;]\!]_{t \,\in\, T_\sigma} && (4.8)\\
& [] \quad [\![\ (\textbf{and}\ (\textbf{eq}\ \mathsf{autState}\ n) \\
& \qquad \mathrm{Cond}(\tau_{event}(\sigma, A.\nu(n))) \rightarrow\ \tau_{event}(\sigma, A.\nu(n))\ ]\!]_{n \,\in\, S_c} && (4.9)\\
& \textbf{fi}
\end{aligned}
$$

The statements generated at Line (4.8) define the execution of transitions t labeled

Table 4.1 – Definition of C_t, T_t, Ω_t, H_t

Type	C_t	T_t	Ω_t	H_t
$\langle \mathsf{loc}, n_1, n_2 \rangle$	C_{loc}	T_{loc}	Ω_{loc}	H
$\langle \mathsf{tsub}, n_1, n_2, n_{2_\flat} \rangle$, with $n_{2_\flat} \notin \{H, H^*\}$	C_{loc}	T_{tosub}	Ω_{tsub}	H
$\langle \mathsf{tsub}, n_1, n_2, H \rangle$	C_{loc}	T_{tosubh}	Ω_{tosubh}	H
$\langle \mathsf{tsub}, n_1, n_2, H^* \rangle$	C_{loc}	T'_{tosubh}	Ω_{tosubh}	H
$\langle \mathsf{fsub}, n_1, n_{1_\flat}, n_2 \rangle$	C_{fsub}	T_{loc}	Ω_{fsub}	H

by σ for one of the first five cases of Fig. 4.4. The statements generated at Line (4.9) define the execution of a transition within a composite state n of the automaton, which corresponds to Case 6 in Fig. 4.4.

The condition C_t and the statements $\langle \Omega_t;\ T_t \rangle$ differ according to the type of the arrow of the transition t (local, to sub-state, to shallow history state H, and to deep history state H*, and from sub-state). The values of C_t, T_t, and Ω_t are computed as follows; the computation of transition guards and final transitions will be described in the subsequent paragraph, as a modification of C_t.

$$
\begin{array}{lcll}
C_{loc} & = & (\textbf{eq}\ \mathsf{autState}\ n_1) & \\
C_{fsub} & = & (\textbf{and}\ (\textbf{eq}\ \mathsf{autState}\ n_1)\ (\textbf{eq}\ \mathsf{cast}(\mathsf{TState\_\$}n_{1},\ \mathsf{ts}).\mathsf{autState}\ n_{1_\flat})) & \\
T_{loc} & = & \mathsf{autState} := n_2;\ \iota(A.\nu(n_2)) & \\
T_{tosub} & = & \mathsf{autState} := n_2;\ \iota(A.\nu(n_2)); & \\
 & & \mathsf{cast}(\mathsf{TState\_\$}n_{2},\ \mathsf{ts}).\mathsf{autState} := n_{2_\flat};\ \iota(A.\nu(n_2).\nu(n_{2_\flat})) & \\
T_{tosubh} & = & \mathsf{autState} := n_2;\ \iota(A.\nu(n_2).\nu(n_{2_\flat})) & \text{with}\ \ n_{2_\flat} = \alpha(h(n_2)) \\
T'_{tosubh} & = & \mathsf{autState} := n_2;\ \mathsf{cast}(\mathsf{TState\_\$}n_{2},\ \mathsf{ts}) := \mathsf{hState}(n_2) & \\
\Omega_{loc} & = & H;\ t.Act_{tr};\ A.\zeta(n_1).Act_{stay};\ A.Act_{astd} & \text{if}\ n_1 = n_2 \\
 & = & H;\ A.\zeta(n_1).Act_{out};\ t.Act_{tr};\ A.\zeta(n_2).Act_{in};\ A.Act_{astd} & \text{else} \\
\Omega_{tsub} & = & H;\ t.Act_{tr};\ A.\zeta(n_1).Act_{stay};\ \iota(A.\nu(n_2)); & \text{if}\ n_1 = n_2 \\
 & & A.\nu(n_2).\zeta(n_{2_\flat}).Act_{in};\ A.\nu(n_2).Act_{astd};\ A.Astd_{astd} & \\
 & = & H;\ A.\zeta(n_1).Act_{out};\ t.Act_{tr};\ A.\zeta(n_2).Act_{in};\ \iota(A.\nu(n_2)); & \text{else} \\
 & & A.\nu(n_2).\zeta(n_{2_\flat}).Act_{in};\ A.\nu(n2).Act_{astd};\ A.Act_{astd} & \\
\Omega_{tsubh} & = & H;\ A.\zeta(n_1).Act_{out};\ t.Act_{tr};\ A.\zeta(n_2).Act_{in}; & \\
 & & A.\nu(n_2).\zeta(n_{2_\flat}).Act_{in};\ A.\nu(n2).Act_{astd};\ A.Act_{astd} & \\
\Omega_{fsub} & = & A.\nu(n_1).\zeta(n_{1_\flat}).Act_{out};\ A.\nu(n_1).Act_{astd};\ H; & \\
 & & A.\zeta(n_1).Act_{out};\ t.Act_{tr};\ A.\zeta(n_2).Act_{in};\ A.Act_{astd} & \\
H & = & \mathsf{hState}(n_1) := \mathsf{ts} &
\end{array}
$$

where $\alpha(\bullet)$ returns the name of the composite state $\bullet$. When t is local, a transition is made from state n_1 to state n_2 when C_{loc} holds. Then, the action $\langle \Omega_{loc};\ T_{loc} \rangle$ is executed. Two cases have to be considered. When $n_1 = n_2$, the stay code $A.\zeta(n_1).Act_{stay}$ should be included in $\langle \Omega_{loc};\ T_{loc} \rangle$ since we loop over state n_1. The global action of $A.Act_{astd}$ is also called on all transitions of A, as the last step. When $n_1 \neq n_2$, we leave state n_1 (triggering the exit code $A.\zeta(n_1).Act_{out}$), execute the transition action $t.Act_{tr}$, and enter in the next state n_2 (triggering the entry code $A.\zeta(n_2).Act_{in}$). The other cases of C_t and Ω_t follow a similar pattern, depending on their semantics defined in the inference rules [1].

The value of C_t must be adjusted when the transition t has a guard, when it is final and to cater for the value of t's parameters, which acts like pattern matching.

1. https://depot.gril.usherbrooke.ca/lionel-tidjon/astd-tech-report26

For the sake of simplicity, we describe these modifications to C_t using the notation

$$C_t \leftarrow (... \ C_t \ ...)$$

When transition t has a guard g, the guard is added to C_t, i.e.,

$$C_t \leftarrow (\textbf{and} \ C_t \ g)$$

When transition t is final, the value $\phi(A.\nu(n_1))$ is added to C_t, i.e.,

$$C_t \leftarrow (\textbf{and} \ C_t \ \phi(A.\nu(n_1)))$$

When transition t is annotated with an event pattern $\sigma(z_1, ..., z_n)$, the condition C_{pat} is added to C_t, i.e.,

$$C_t \leftarrow (\textbf{and} \ C_t \ C_{pat})$$

where $C_{pat} = (\textbf{and} \ C_{p_1,z_1} \ C_{p_2,z_2} \ .... \ C_{p_n,z_n})$, p_i is the parameter of the function processing event σ. z_i can be one of the following:

1. an expression constructed using identifiers available in that scope, or constants. The equality $C_{p_i,z_i} = (\textbf{eq} \ p_i \ z_i)$ is generated.
2. $?x : T$: this declares a new variable x whose scope is just the transition t. In that case, no condition is generated; the occurrences of x elsewhere in the transition are replaced with p_i (i.e., in the guard and in the action).

Example. Let us consider the automaton ASTDs D and E (see Fig. 4.2). The translation of variables and types is computed as follows,

$$
\begin{aligned}
\tau_{type}(D) \triangleq\ & \textbf{def type } \mathsf{TState\_}D \ = \ \textbf{record } \{ \\
& \quad z1 : int, \\
& \quad z2 : int, \\
& \quad \mathsf{autState} : \mathsf{AutState} \\
& \}; \\
& \textbf{const } \mathsf{shallow\_final\_}D : \mathsf{Set{<}AutState{>}} = \{2\}; \\
\tau_{type}(E) \triangleq\ & \textbf{def type } \mathsf{TState\_}E \ = \ \textbf{record } \{ \\
& \quad z3 : int, \\
& \quad \mathsf{autState} : \mathsf{AutState} \\
& \};
\end{aligned}
$$

where $\mathsf{AutState} = \{1, 2, 3\}$. The event a appears on local transitions $1 \to 2$ and $3 \to 3$. There is no history state in TState_D because D has none. The body of the event function $a(p1 : int)$ in D is defined by

$$\tau_a(D) \triangleq \textbf{if } C_{1\to 2} \ \to \ \Omega_{1\to 2} \ ; \ T_{1\to 2} \textbf{ fi}$$

where

$$
\begin{aligned}
C_{1\to 2} &= (\textbf{and } (\textbf{eq } \mathsf{autState}\ 1)\ (\textbf{and } (u < 10)\ (\textbf{eq } p1\ u))) \\
\Omega_{1\to 2} &= \{z2 := z2 + 1;\ z1 := z1 + 1\} \\
T_{1\to 2} &= \{\mathsf{autState} := 2\}
\end{aligned}
$$

The body of $a(p1 : int)$ in E is given by

$$\tau_a(E) \triangleq \textbf{if } C_{3\to 3} \ \to \ \Omega_{3\to 3} \ ; \ T_{3\to 3} \textbf{ fi}$$

For the transition $3 \rightarrow 3$, we have

$$C_{3\rightarrow 3} = (\textbf{and}\ (\textbf{eq}\ (\mathsf{autState}\ 2)\ (u >= 10))$$
$$\Omega_{3\rightarrow 3} = \{z3 := z3 + 1\}$$
$$T_{3\rightarrow 3} = \{\mathsf{autState} := 2\}$$

Kleene ASTD. Let A be a Kleene ASTD [310]. Recall that A has the following structure:

$$\langle \star, b \rangle$$

where $b \in \mathsf{ASTD}$ is the body of the closure. The type of a closure state is $\langle \star_\circ, E, started?, s \rangle$ where $s \in \mathsf{States}$, E denotes the attribute values of the closure ASTD, and $started? \in \mathsf{Boolean}$ indicates whether the first iteration has been started.

State and Attributes. The types generated for kleene A are defined as follows:

$$\begin{aligned}
\tau_{type}(A) \triangleq\ & \textbf{def type}\ \mathsf{TState_\$}A\ =\ \textbf{record}\ \{ \\
& [\![,\ \$v.name : \$v.T]\!]_{v\ \in\ A.Attr} \\
& \mathsf{kleeneState} : \mathsf{KleeneState}, \\
& \mathsf{ts_\$}b : \mathsf{TState_\$}b \\
& \}; \\
& \tau_{type}(A.b)
\end{aligned}$$

where $\mathsf{TState\_\$}A$ holds the attributes of A and the state of the body $A.b$. The variable $\mathsf{kleeneState}$ of type $\mathsf{KleeneState} = \{\mathsf{started},\ \mathsf{notstarted}\}$ stores the current state of A. $\tau_{type}(A.b)$ returns the translation of types and attributes of the body $A.b$.
$\iota(A)$ and $\phi(A)$ are computed as follows,

$$\iota(A) \triangleq \textsf{kleeneState} := \textsf{notstarted};$$
$$[\![; \$v.name := \$v.init]\!]_{v \in A.Attr}$$
$$\iota(A.b)$$
$$\phi(A) \triangleq (\textbf{or } (\textbf{eq } \textsf{kleeneState } \textsf{notstarted}) \ \phi(A.b))$$

After calling $\iota(A)$, the variable kleeneState has an initial value notstarted; it represents the case when A is not started yet.

Event. The semantics of ASTDs [310] often requires to check that a transition from the initial state of an ASTD can be fired, for example in a Kleene closure, where a new iteration can be started when the body of the closure is in a final state. To do so, we need the ability to substitute, in an expression, the occurrences of the state variables and the attributes of A with their values in the initial state. For that purpose, we define the substitution operator $\mathcal{S}_{init}(A,\ E)$ as follows

$$\mathcal{S}_{init}(A,\ E) \triangleq [\iota(A)]\ E$$

such that

$$[\iota(A)] = [v_1 := \ i_1;\ ...;\ v_n := \ i_n]$$

where $[\iota(A)]$ is a composition of substitutions representing the sequential execution of the assignment statements of the initialisation. $[\iota(A)]$ substitutes the states and attributes of A (i.e., $v_1, ..., v_n$) by their initial values (i.e., $i_1, ..., i_n$).
When $\sigma \in \Sigma_A$, $\tau_{event}(\sigma, A)$ has the form,

$$\tau_{event}(\sigma, A) \triangleq$$
$$\textbf{if } C_1 \rightarrow Act_1$$
$$[]\ C_2 \ \rightarrow \ Act_2$$
$$\textbf{fi}$$

The values of C_i and Act_i are computed as follows,

$$
\begin{aligned}
C_1 &= (\textbf{and}\ \phi(A.b)\ \mathcal{S}_{init}(A.b,\ Cond(\tau_{event}(\sigma, A.b)))\ ) \\
C_2 &= Cond(\tau_{event}(\sigma, A.b)) \\
Act_1 &= \mathsf{kleeneState} := \mathsf{started};\ \mathcal{S}_{init}(A.b,\ \langle \tau_{event}(\sigma, A.b)\rangle);\ A.Act_{astd} \\
Act_2 &= \mathsf{kleeneState} := \mathsf{started};\ \tau_{event}(\sigma, A.b);\ A.Act_{astd}
\end{aligned}
$$

There are two inference rules to consider when triggering a transition in a Kleene closure. Condition C_1 denotes the first inference rule: when sub-ASTD $A.b$ is in a final state, it can trigger a transition from its initial state. Thus, condition $\mathcal{S}_{init}(A.b,$ $Cond(\tau_{event}(\sigma, A.b)))$ checks that the initial state of $A.b$ can execute event σ by taking the condition generated for executing sigma in $A.b$ and substituting the values of the attributes and the state variables with their value in the initial state. The second inference rule is represented by condition C_2: the transition is triggered from the current state of the sub-ASTD.

When C_1 holds, the action Act_1 updates the state of A to started, initializes and executes the sub-ASTD $A.b$. The condition C_2 checks if at least one condition holds in the body (i.e., $\tau_{event}(\sigma, A.b)$) before executing it.

Example. Let us consider the Kleene ASTD C in the example. The translation of variables and types is computed as follows,

$$
\begin{aligned}
\tau_{type}(C) \triangleq\ & \textbf{def type}\ \mathsf{TState\_}C\ =\ \textbf{record \{} \\
& \quad y : int, \\
& \quad \mathsf{kleeneState} : \mathsf{KleeneState}, \\
& \quad \mathsf{ts\_}D : \mathsf{TState\_}D, \\
& \textbf{\}}; \\
& ...
\end{aligned}
$$

ASTD C contains the automaton D. Following translation rule, the body of the event function $a(p1 : int)$ in C is defined by,

$$
\begin{array}{l}
\tau_a(C) \triangleq \\
\quad \textbf{if } (\textbf{and } \phi(C.D)\ \mathcal{S}_{init}(C.D,\ Cond(\tau_{event}(a, C.D)))) \rightarrow \\
\quad\quad \mathsf{kleeneState} := \mathsf{started};\ \mathcal{S}_{init}(C.D,\ \langle \tau_{event}(a, C.D) \rangle);\ y := y + 1 \\
\quad [] \ Cond(\tau_{event}(a, C.D)) \rightarrow\ \mathsf{kleeneState} := \mathsf{started};\ \tau_{event}(a, C.D);\ y := y + 1 \\
\quad \textbf{fi}
\end{array}
$$

where

$$
\begin{array}{l}
\tau_{event}(a, C.D) \triangleq \textbf{if } (\textbf{and } (\textbf{eq } \mathsf{autState}\ 1)\ (\textbf{and } (u < 10)\ (\textbf{eq } p1\ u))) \rightarrow \\
\qquad\qquad z2 := z2 + 1;\ z1 := z1 + 1 \\
\qquad \textbf{fi}
\end{array}
$$

$$
\begin{array}{rl}
\phi(C.D) & \triangleq (\textbf{in } \mathsf{autState}\ \mathsf{shallow\_final\_}D) \\
Cond(\tau_{event}(a, C.D)) & \triangleq (\textbf{and } (\textbf{eq } \mathsf{autState}\ 1)\ (\textbf{and } (u < 10)\ (\textbf{eq } p1\ u))) \\
\mathcal{S}_{init}(C.D,\ Cond(\tau_{event}(a, C.D))) & \triangleq (\textbf{and } (\textbf{eq } 1\ 1)\ (\textbf{and } (u < 10)\ (\textbf{eq } p1\ u)))
\end{array}
$$

The substitution $\mathcal{S}_{init}(C.D,\ \langle \tau_{event}(a, C.D) \rangle)$ results in the following code,

$$
\begin{array}{l}
\textbf{if } (\textbf{and } (\textbf{eq } 1\ 1)\ (\textbf{and } (u < 10)\ (\textbf{eq } p1\ u))) \rightarrow \\
\quad z2 := 0 + 1;\ z1 := 0 + 1 \\
\textbf{fi}
\end{array}
$$

Parameterized synchronization ASTD. Let A be a parameterized synchronization ASTD [310]. Recall that A has the following structure:

$$\langle |[]|, \Delta, l, r \rangle$$

where Δ is the synchronization set of event labels, $l, r \in \mathsf{ASTD}$ are the synchronized ASTDs. When the label of the event to execute belongs to Δ, the two sub-ASTDs must both execute it; otherwise either the left or the right sub-ASTD can execute it. When $\Delta = \varnothing$, the synchronization is called an *interleaving* and is abbreviated as $|||$. The state of A is of type $\langle |[]|_{\circ}, E, s_l, s_r \rangle$, where s_l, s_r are the states of the left and right sub-ASTDs.

State and Attributes. The types generated for parameterized synchronization A are defined as follows:

$$\begin{aligned}
\tau_{type}(A) \triangleq\ & \textbf{def type } \mathsf{TState_}\$A \ = \ \textbf{record } \{ \\
& [\![,\ \$v.name : \$v.T]\!]_{v\ \in\ A.Attr} \\
& \mathsf{ts_}\$l : \mathsf{TState_}\$l, \\
& \mathsf{ts_}\$r : \mathsf{TState_}\$r \\
& \}; \\
& \tau_{type}(A.l); \\
& \tau_{type}(A.r)
\end{aligned}$$

where $\mathsf{TState\_}\$A$ holds the attributes of A, and the state of l and r. $\tau_{type}(A.l)$ and $\tau_{type}(A.r)$ return the translation of $A.l$ and $A.r$, respectively. For a parameterized synchronization ASTD, $\iota(A)$ and $\phi(A)$ are computed as follows,

$$\begin{aligned}\iota(A) \triangleq\ & [\![;\ \$v.name\ :=\ \$v.init]\!]_{v\ \in\ A.Attr} \\ & \iota(A.l); \\ & \iota(A.r) \\ \phi(A) \triangleq\ & (\textbf{and}\ \phi(A.l)\ \phi(A.r))\end{aligned}$$

The initial state of A is composed of the initial states of l and r. A is final if l and r are finals.

Event. Let σ an event that appears in $A.l$, $A.r$, or both. Let us consider first the case where $\sigma \notin \Delta$.
If σ only appears on the left side, we have

$$\tau_{event}(\sigma, A)\ \triangleq\ \textbf{if}\ Cond(\tau_{event}(\sigma, A.l))\ \rightarrow\ \tau_{event}(\sigma, A.l);\ A.Act_{astd}\ \textbf{fi}$$

If σ only appears on the right side, we have

$$\tau_{event}(\sigma, A)\ \triangleq\ \textbf{if}\ Cond(\tau_{event}(\sigma, A.r))\ \rightarrow\ \tau_{event}(\sigma, A.r);\ A.Act_{astd}\ \textbf{fi}$$

This means that transitions in the left and right side are executing in interleaving. If σ appears on both side; the body of σ is computed as follows,

$$\begin{aligned}\tau_{event}(\sigma, A) \triangleq\ & \\ & \textbf{if}\ Cond(\tau_{event}(\sigma, A.l))\ \rightarrow\ \tau_{event}(\sigma, A.l);\ A.Act_{astd} \\ & []\ Cond(\tau_{event}(\sigma, A.r))\ \rightarrow\ \tau_{event}(\sigma, A.r);\ A.Act_{astd} \\ & \textbf{fi}\end{aligned}$$

When both sides can execute σ, a non-deterministic choice is made between the two. When $\sigma \in \Delta$, both sides must synchronize; if they both can execute it, then their body are executed in sequence. The semantics of ASTDs requires that the actions

executed by each side must be commutative, that is, the final value of the state variables declared in A, which can be modified by both sides, is the same, whether $A.l$ or $A.r$ is executed first. Thus, the implementation is free to choose the order of execution. Here, we (arbitrarily) start with $A.l$. The operation $\tau_{event}(\sigma, A)$ is expressed as follows,

$$\begin{aligned}\tau_{event}(\sigma, A) \triangleq\ & \\ & \textbf{if } (\textbf{and } Cond(\tau_{event}(\sigma, A.l))\ Cond(\tau_{event}(\sigma, A.r))) \rightarrow \\ & \qquad \tau_{event}(\sigma, A.l);\ \tau_{event}(\sigma, A.r);\ A.Act_{astd} \\ & \textbf{fi}\end{aligned}$$

Quantified synchronization ASTD. Let A be a quantified synchronization ASTD [310]. Recall that A has the following structure:

$$\langle |[]|:, x, T, \Delta, b \rangle$$

where $x \in \mathsf{Var}$ is a quantified variable, T is a set of values, $\Delta \subseteq \mathsf{Label}$ is a set of event labels on which synchronization must occur, and $b \in \mathsf{ASTD}$ is the sub-ASTD of the synchronization. When $\Delta = \varnothing$, A is called a quantified interleaving. The state of A is of type $\langle |[]|:_\circ, E, f \rangle$ where $|[]|:_\circ$ is the constructor, E the values of attributes and $f \in T \rightarrow \mathsf{States}$ is a function which associates a state of b to each value of T.

State and Attributes. The types generated for quantified synchronization A are defined as follows:

$$
\begin{aligned}
\tau_{type}(A) \triangleq\ & \textbf{def type } \mathsf{TState\_}\$A \ = \ \textbf{record } \{ \\
& \quad [\![,\ \$v.name : \$v.T]\!]_{v\ \in\ A.Attr} \\
& \quad x : A.x.T, \\
& \quad f : \mathsf{Map}\langle A.x.T,\ \mathsf{TState\_}\$b\rangle \\
& \}; \\
& \textbf{const } \mathsf{T\_}\$A : \mathsf{Set}\langle A.x.T\rangle = A.T.values; \\
& \tau_{type}(A.b)
\end{aligned}
$$

where $\mathsf{TState\_}\$A$ holds the attributes of A, the quantified variable x, and the map f that stores the state of $A.b$ for each value in $A.x.T$. $A.x.T$ is the type of the quantified variable x. $\mathsf{T\_}\$A$ is a constant set of type $\mathsf{Set}\langle A.x.T\rangle$; it is used when T is an enumerated set. The initialisation operation $\iota(A)$ is computed as follows,

$$\iota(A) = [\![;\ \$v.name\ :=\ \$v.init]\!]_{v\ \in\ A.Attr}$$

f is initialized to an empty map. The final operation $\phi(A)$ is expressed as follows,

$$\phi(A) = \mathsf{forall}(\mathsf{T\_}\$A,\ \phi(A.b),\ f)$$

where operation $\mathsf{forall}(S, C, g) = \forall\ s \in S\ [fields(\mathsf{TState}A.b) := g(s)]\ C$. We use the forall construct to check that final holds for each instance of $A.b$; the state of these instances is stored in f; the quantifier $\forall\, s \in S$ is implemented by a loop over S. The operator $fields(\bullet)$ returns all the field names of $\bullet$. The initialisation is optimised by initializing the map to empty and by adding instances only when they are created (or needed) when an event is received for that instance. To evaluate if a quantified synchronization is in a final state, we can simply evaluate ϕ for the instances created in f; all the others are in the initial state; they can be evaluated all at once using $\phi(\iota(A.b))$, since it does not depend on the value of x.

Event. Let σ be an event that appears in $A.b$. When $\sigma \notin \Delta$, the translation of the body of σ is computed as follows,

$$
\begin{aligned}
&\tau_{event}(\sigma, A) \triangleq \\
&\qquad \textbf{if } \mathsf{exists}(x,\ \mathsf{T\_\$}A,\ Cond(\tau_{event}(\sigma, A.b)),\ f)\ \rightarrow \\
&\qquad\quad [fields(\mathsf{TState\_\$}A.b)\ :=\ f(x)]\ \tau_{event}(\sigma, A.b); \\
&\qquad\quad A.Act_{astd} \\
&\qquad \textbf{fi}
\end{aligned}
$$

where $\mathsf{exists}(u, S, C, g)$ searches for a value $s \in S$ such that $([fields(\mathsf{TState\_\$}name)\ := g(s)]\ C)$ holds and sets u to s when it does hold. The *exists* condition can be optimized when the value of the quantified variable can be determined by the constraints imposed by the transitions in the automata of $A.b$. This process is called κ-optimization and is defined in [95] for EB3 process expression; it has been adapted here for ASTD expressions. When $\mathsf{T\_\$}A$ is not an enumerated set and $A.b$ is not κ-optimizable, then the exists condition cannot be implemented, because the number of values is unbounded or too large to be executed (e.g.,string, int).
When $\sigma \in \Delta$,the translation of the body of σ is computed as follows,

$$
\begin{aligned}
&\tau_{event}(\sigma, A) \triangleq \\
&\qquad \textbf{if } \mathsf{forall}(x,\ \mathsf{T\_\$}A,\ Cond(\tau_{event}(\sigma, A.b)),\ f)\ \rightarrow \\
&\qquad\quad \textbf{for } (\textbf{in } x\ \mathsf{T\_\$}A) \\
&\qquad\qquad [fields(\mathsf{TState\_\$}A.b)\ :=\ f(x)]\ \tau_{event}(\sigma, A.b); \\
&\qquad\quad \textbf{endfor} \\
&\qquad\quad A.Act_{astd} \\
&\qquad \textbf{fi}
\end{aligned}
$$

In this case, we execute $\tau_{event}(\sigma, A.b)$ for each state of $A.b$ stored in f. Finally, we execute the ASTD action $A.Act_{astd}$.

Example. Let us consider the quantified interleaving B from Fig. 4.2. The set of values T_B is {1,2,3,4}. The quantified variable u is applied on the Kleene ASTD C. The translation of variables and types is computed as follows,

$$
\begin{aligned}
\tau_{type}(B) \triangleq\ & \textbf{def type } \mathsf{TState}\_B \ = \ \textbf{record \{} \\
& \quad x : int, \\
& \quad u : int, \\
& \quad f : \mathsf{Map}\langle int, \mathsf{TState}\_C \rangle \\
& \textbf{\};} \\
& \textbf{const } \mathsf{T}\_\$B : \mathsf{Set}\langle int \rangle = \{1, 2, 3, 4\}; \\
& \ldots
\end{aligned}
$$

Following the translation rules, the body of $a(p1 : int)$ in B is given by

$$
\begin{aligned}
\tau_a(B) \triangleq\ & \\
& \textbf{if } \mathsf{exists}(u,\ T\_B, Cond(\tau_{event}(a, B.C)), f) \ \rightarrow \\
& \quad [fields(\mathsf{TState}\_\$B.C) := f(u)]\ \tau_{event}(a, B.C); \\
& \quad x := x + 1 \\
& \textbf{fi}
\end{aligned}
$$

where

$$\mathsf{exists}(u,\ T\_B, Cond(\tau_{event}(a, B.C)),\ f) \triangleq$$
$$\begin{aligned}
&\textbf{for } (\textbf{in } t\ T_B)\\
&\quad u := t\\
&\quad \textbf{if } Cond(\tau_{event}(a, B.C))\\
&\qquad \rightarrow\ \mathsf{return}(\mathsf{true})\\
&\quad \textbf{fi}\\
&\textbf{endfor}\\
&\mathsf{return}(\mathsf{false})
\end{aligned}$$

with

$$Cond(\tau_{event}(a, B.C)) \triangleq\ (\textbf{or}\ \ (\textbf{and}\ \phi(C.D)\ \mathcal{S}_{init}(C.D,\ \ Cond(\tau_{event}(a, C.D))))$$
$$Cond(\tau_{event}(a, C.D))\ )$$

$$\begin{aligned}
\phi(C.D) &= (\textbf{in}\ f(u).\mathsf{ts_}C.\mathsf{ts_}D.\mathsf{autState}\ f(u).\mathsf{ts_}C.\mathsf{ts_}D.\mathsf{shallow_final_}D)\\
Cond(\tau_{event}(a, C.D)) &= (\textbf{and}\ (\textbf{eq}\ f(u).\mathsf{ts_}C.\mathsf{ts_}D.\mathsf{autState}\ 1)\ (\textbf{and}\ (u < 10)\ (\textbf{eq}\ p1\ u)))\\
\mathcal{S}_{init}(C.D,\ \ Cond(\tau_{event}(a, C.D))) &= (\textbf{and}\ (\textbf{eq}\ 1\ 1)\ (\textbf{and}\ (u < 10)\ (\textbf{eq}\ p1\ u))))
\end{aligned}$$

Likewise, the substitution of the action

$$
\begin{aligned}
[fields(\mathsf{TState}B.C) := f(u)]\tau_{event}(a, B.C) \triangleq\ & \textbf{if } (\textbf{and } \phi(C.D)\ \mathcal{S}_{init}(C.D,\ Cond(\tau_{event}(a, C.D)))) \rightarrow \\
& \quad f(u).\mathsf{ts\_}C.\mathsf{kleeneState} := \mathsf{started}; \\
& \quad \mathcal{S}_{init}(C.D,\ \langle \tau_{event}(a, C.D); \\
& \qquad f(u).\mathsf{ts\_}C.y := f(u).\mathsf{ts\_}C.y + 1\rangle) \\
& [] \ Cond(\tau_{event}(a, C.D)) \rightarrow \\
& \quad f(u).\mathsf{ts\_}C.\mathsf{kleeneState} := \mathsf{started};\ \tau_{event}(a, C.D); \\
& \quad f(u).\mathsf{ts\_}C.y := f(u).\mathsf{ts\_}C.y + 1 \\
& \textbf{fi} \\
& x := x + 1
\end{aligned}
$$

where

$$
\begin{aligned}
\tau_{event}(a, C.D) \triangleq\ & \textbf{if } (\textbf{and } (\textbf{eq } f(u).\mathsf{ts\_}C.\mathsf{ts\_}D.\mathsf{autState}\ 1)\ (\textbf{and } (u < 10)\ (\textbf{eq } p1\ u))) \rightarrow \\
& \quad f(u).\mathsf{ts\_}C.\mathsf{ts\_}D.z2 := f(u).\mathsf{ts\_}C.\mathsf{ts\_}D.z2 + 1; \\
& \quad f(u).\mathsf{ts\_}C.\mathsf{ts\_}D.z1 := f(u).\mathsf{ts\_}C.\mathsf{ts\_}D.z1 + 1 \\
& \textbf{fi}
\end{aligned}
$$

and

$$
\begin{aligned}
\mathcal{S}_{init}(C.D,\ \langle \tau_{event}(a, C.D);\ f(u).\mathsf{ts\_}C.y := f(u).\mathsf{ts\_}C.y + 1\rangle) \triangleq\ & \\
& \textbf{if } (\textbf{and } (\textbf{eq } 1\ 1) \\
& \qquad (\textbf{and } (u < 10)\ (\textbf{eq } p1\ u))) \rightarrow \\
& \quad f(u).\mathsf{ts\_}C.\mathsf{ts\_}D.z2 := 0 + 1; \\
& \quad f(u).\mathsf{ts\_}C.\mathsf{ts\_}D.z1 := 0 + 1 \\
& \textbf{fi} \\
& f(u).\mathsf{ts\_}C.y := f(u).\mathsf{ts\_}C.y + 1
\end{aligned}
$$

Flow ASTD. A flow ASTD A has structure $\langle \Cup, l, r\rangle$, where l and r are left and right components of A. The state of A is of type $\langle \Cup_\circ, E, s_l, s_r\rangle$, where s_l, s_r are the states of the left and right sub-ASTDs.

State and Attributes. The types generated for flow A are defined as follows:

$$\begin{aligned}\tau_{type}(A) \triangleq\ & \textbf{def type } \mathsf{TState\_}\$A \ = \ \textbf{record \{} \\ & [\![,\ \$v.name : \$v.T]\!]_{v\ \in\ A.Attr} \\ & \mathsf{ts\_}\$l : \mathsf{TState\_}\$l, \\ & \mathsf{ts\_}\$r : \mathsf{TState\_}\$r \\ & \textbf{\}}; \\ & \tau_{type}(A.l); \\ & \tau_{type}(A.r)\end{aligned}$$

where $\mathsf{TState\_}\$A$ holds the attributes of A and the states of l and r. $\tau_{type}(A.l)$ and $\tau_{type}(A.r)$ return the translation of $A.l$ and $A.r$, respectively.
For a flow ASTD, $\iota(A)$ and $\phi(A)$ are computed as follows,

$$\begin{aligned}\iota(A) \triangleq\ & [\![;\ \$v.name \ := \ \$v.init]\!]_{v\ \in\ A.Attr} \\ & \iota(A.l); \\ & \iota(A.r) \\ \phi(A) \triangleq\ & (\textbf{and } \phi(A.l)\ \phi(A.r))\end{aligned}$$

Event. Let σ an event that can appear in $A.l$, $A.r$, or both. The value of $\tau_{event}(\sigma, A)$ is computed as follows,

$$
\begin{aligned}
\tau_{event}(\sigma, A) \triangleq\ & \\
& \textbf{var b} : \text{Boolean} = \text{false}; \\
& \textbf{if } \mathit{Cond}(\tau_{event}(\sigma, A.l)) \rightarrow \\
& \quad \tau_{event}(\sigma, A.l); \\
& \quad \text{b} := \text{true}; \\
& \textbf{fi} \\
& \textbf{if } \mathit{Cond}(\tau_{event}(\sigma, A.r)) \rightarrow \\
& \quad \tau_{event}(\sigma, A.r); \\
& \quad \text{b} := \text{true}; \\
& \textbf{fi} \\
& \textbf{if } (\textbf{eq b true}) \rightarrow A.Act_{astd}; \textbf{ fi}
\end{aligned}
$$

The left side and the right side execute independently of one another. The semantics requires that if they both modify the attributes of the flow ASTD, then their modification must be commutative, as in the synchronization ASTD. The global condition *Cond* (to be used by parent ASTDs) is the disjunction of $\mathit{Cond}(\tau_{event}(\sigma, A.l))$ and $\mathit{Cond}(\tau_{event}(\sigma, A.r))$.

Example. Let us consider the flow ASTD A in Fig. 4.2. The translation of variables and types is computed as follows,

$$
\begin{aligned}
\tau_{type}(A) \triangleq\ & \textbf{def type } \text{TState\_}A = \textbf{record \{} \\
& \quad \text{ts\_}B : \text{TState\_}B, \\
& \quad \text{ts\_}E : \text{TState\_}E \\
& \textbf{\}}; \\
& \ldots \\
& \text{ts\_}A : \text{TState\_}A
\end{aligned}
$$

Event a appears on local transitions $1 \rightarrow 2$ and $3 \rightarrow 3$. The translation of the body of $a(p1 : int)$ is given by,

$$
\begin{aligned}
&\tau_{event}(a, A) \triangleq \\
&\qquad \textbf{var b} \ : \ \text{Boolean} \ = \ \text{false}; \\
&\qquad \textbf{if exists}(u, \ T, Cond(\tau_{event}(a, A.B.C)), f) \ \rightarrow \\
&\qquad\quad \tau_{event}(a, A.B); \\
&\qquad\quad \text{b} := \text{true} \\
&\qquad \textbf{if} \\
&\qquad \textbf{if } (\textbf{and } (\textbf{eq } (\text{autState } 2) \ (u >= 10)) \ \rightarrow \\
&\qquad\quad \tau_{event}(\sigma, A.E); \\
&\qquad\quad \text{b} := \text{true} \\
&\qquad \textbf{if} \\
&\qquad \textbf{if } (\textbf{eq b true}) \ \rightarrow A.Act_{astd} \ \textbf{fi}
\end{aligned}
$$

where $A.Act_{astd} = \varnothing$. The translation operations $\tau_{event}(a, A.B)$ and $\tau_{event}(\sigma, A.E)$ were previously computed. We can observe that some conditions of **if-fi** statements are re-evaluated several times due to the nesting of conditions; thus a pruning operation is necessary to optimize the generated code. This will be discussed later in Sect. 4.3.3.

Sequence ASTD. Let A be a sequence ASTD [310] that has the following structure:

$$\langle \leftrightarrows, fst, snd \rangle$$

where *fst* and *snd* are ASTDs denoting respectively the first and second sub-ASTDs of the sequence. A sequence state is of type $\langle \leftrightarrows_{\circ}, E, [\text{fst} \mid \text{snd}], s \rangle$, where $\leftrightarrows_{\circ}$ is a constructor of the sequence state, E the values of attributes declared in the sequence, [fst | snd] is a choice between two markers that respectively indicate whether the sequence is in the first sub-ASTD or the second sub-ASTD and $s \in \text{States}$.

State and Attributes. The types generated for sequence A are defined as follows:

$$
\begin{aligned}
\tau_{type}(A) \triangleq\ & \textbf{def type } \mathsf{TState\_\$}A \ = \ \textbf{record } \{ \\
& [\![\ \$v.name : \$v.T]\!]_{v \ \in \ A.Attr} \\
& \mathsf{sequenceState} : \mathsf{SequenceState}, \\
& \mathsf{ts} : \mathsf{TState\_\$fst} \ \oplus \ \mathsf{TState\_\$snd} \\
& \}; \\
& \tau_{type}(A.\mathsf{fst}); \\
& \tau_{type}(A.\mathsf{snd})
\end{aligned}
$$

where $\mathsf{TState\_\$}A$ holds the attributes of A, the state of A, and the state of fst and snd. The variable sequenceState of type $\mathsf{SequenceState} = \{\mathsf{fst}, \ \mathsf{snd}\}$ stores the current state of A. The field ts is a sum type, i.e., $\mathsf{TState\_\$fst}$ and $\mathsf{TState\_\$snd}$. $\tau_{type}(A.\mathsf{fst})$ and $\tau_{type}(A.\mathsf{snd})$ return the translation of attributes and types of $A.\mathsf{fst}$ and $A.\mathsf{snd}$.
For an ASTD sequence, $\iota(A)$ and $\phi(A)$ are computed as follows,

$$
\begin{aligned}
\iota(A) \triangleq\ & \mathsf{sequenceState} \ := \ \mathsf{fst}; \\
& [\![; \ \$v.name \ := \ \$v.init]\!]_{v \ \in \ A.Attr} \\
& \iota(A.\mathsf{fst}) \\
\phi(A) \triangleq\ & (\textbf{or } (\textbf{and } (\textbf{eq } \mathsf{sequenceState}\ \mathsf{fst}) \\
& \qquad (\textbf{and } \phi(A.\mathsf{fst})\ \mathcal{S}_{init}(A.\mathsf{snd}, \ \phi(A.\mathsf{snd})))) \\
& \quad (\textbf{and } (\textbf{eq } \mathsf{sequenceState}\ \mathsf{snd})\ \phi(A.\mathsf{snd})))
\end{aligned}
$$

Event. A sequence is final when either it is executing the first ASTD, its state is final state, and the initial state of the second ASTD is final, or when it is executing the second ASTD and its state is final. Let σ be an event that appears in $A.fst$ or $A.snd$. For a sequence ASTD, the body of σ in the IL language is defined by

$$\tau_{event}(\sigma, A) \triangleq$$
$$\begin{array}{lll} \textbf{if} & C_1 \rightarrow Act_1 & \text{if } \sigma \text{ appears in } A.\mathsf{fst} \\ [] & C_2 \rightarrow Act_2 & \left.\begin{array}{l}\\\\\end{array}\right\} \text{if } \sigma \text{ appears in } A.\mathsf{snd} \\ [] & C_3 \rightarrow Act_3 & \\ \textbf{fi} & & \end{array}$$

The values of C_i and Act_i are computed as follows,

$$C_1 = (\textbf{and } (\textbf{eq } \mathsf{sequenceState}\ \mathsf{fst})\ Cond(\tau_{event}(\sigma, A.\mathsf{fst})))$$

$$C_2 = (\textbf{and } (\textbf{eq } \mathsf{sequenceState}\ \mathsf{fst}) \\ (\textbf{and } \phi(A.\mathsf{fst})\ \mathcal{S}_{init}(A.\mathsf{snd},\ Cond(\tau_{event}(\sigma, A.\mathsf{snd})))))$$

$$C_3 = (\textbf{and } (\textbf{eq } \mathsf{sequenceState}\ \mathsf{snd})\ Cond(\tau_{event}(\sigma, A.\mathsf{snd})))$$
$$Act_1 = \tau_{event}(\sigma, A.\mathsf{fst});\ A.Act_{astd}$$
$$Act_2 = \mathsf{sequenceState} := \mathsf{snd};\ \mathcal{S}_{init}(A.\mathsf{snd},\ \langle\tau_{event}(\sigma, A.\mathsf{snd})\rangle);\ A.Act_{astd}$$
$$Act_3 = \tau_{event}(\sigma, A.\mathsf{snd});\ A.Act_{astd}$$

The condition C_1 checks if the state of A is fst and at least one condition is true in $\tau_{event}(\sigma, A.\mathsf{fst})$. When C_1 is true, the first component is executed (i.e., Act_1). The condition C_2 checks if the first component is in final state and if the second ASTD can execute σ from its initial state. When it is the case, Act_2 updates the state of A to snd and initializes the second component. Finally, the condition C_3 checks if at least one condition is true in $\tau_{event}(\sigma, A.\mathsf{snd})$ before executing it.

Choice ASTD. Let A be a choice ASTD [310] with the following structure:

$$\langle|, l, r\rangle$$

where $l, r \in$ ASTD are respectively the first and second element of the choice. The type of a choice state is $\langle|_\circ, E, side, s\rangle$ where $side \in (\bot \mid \langle\mathsf{left}\rangle \mid \langle\mathsf{right}\rangle)$ denotes the sub-ASTD which has been chosen, $s \in (\mathsf{States} \mid \bot)$ denotes the state of the sub-ASTD which has been chosen and E the values of attributes declared in the choice ASTD.

State and Attributes. The types generated for choice A are defined as follows:

$$
\begin{aligned}
\tau_{type}(A) \triangleq\ & \textbf{def type } \mathsf{TState\_}\$A \ = \ \textbf{record } \{ \\
& \quad [\![\ \$v.name : \$v.T]\!]_{v \ \in \ A.Attr} \\
& \quad \mathsf{choiceState} : \mathsf{ChoiceState}, \\
& \quad \mathsf{ts} : \mathsf{TState\_}\$l \ \oplus \ \mathsf{TState\_}\$r \\
& \} ; \\
& \tau_{type}(A.l); \\
& \tau_{type}(A.r)
\end{aligned}
$$

where $\mathsf{TState\_}\$A$ holds the attributes of A, the state of A, and the state of l and r. The variable choiceState, of type $\mathsf{ChoiceState} = \{\mathsf{none}, \ \mathsf{left}, \ \mathsf{right}\}$, stores the current state of A. The structure ts is a sum type, i.e., it can be of any of type $\mathsf{TState\_}\$l$ and $\mathsf{TState\_}\$r$. $\tau_{type}(A.l)$ and $\tau_{type}(A.r)$ return the translation of $A.l$ and $A.r$, respectively. For a choice ASTD, $\iota(A)$ and $\phi(A)$ are computed as follows,

$$
\begin{aligned}
\iota(A) = \ & \mathsf{choiceState} \ := \ \mathsf{none}; \\
& [\![; \ \$v.name \ := \ \$v.init]\!]_{v \ \in \ A.Attr} \\
\phi(A) = \ & (\textbf{or } (\textbf{and } (\textbf{eq } \mathsf{choiceState } \mathsf{none}) \ (\textbf{or } \mathcal{S}_{init}(A.l, \ \phi(A.l)) \ \mathcal{S}_{init}(A.r, \ \phi(A.r)))) \\
& \quad (\textbf{or } (\textbf{and } (\textbf{eq } \mathsf{choiceState } \mathsf{left}) \ \phi(A.l)) \\
& \quad\quad (\textbf{and } (\textbf{eq } \mathsf{choiceState } \mathsf{right}) \ \phi(A.r))))
\end{aligned}
$$

The variable choiceState has initial value none; it means that no choice has been made. It takes value left (resp. right) if the left side (resp. left) of A has been chosen.

Event. Let σ an event that appears in $A.l$ or $A.r$. For a choice ASTD, the body of σ in the IL language is defined by

$$\begin{array}{lll} \tau_{event}(\sigma, A) & \triangleq & \\ & \left.\begin{array}{l} \textbf{if}\ \ C_1 \ \rightarrow\ Act_1 \\ []\ \ C_2 \ \rightarrow\ Act_2 \end{array}\right\} & \text{if } \sigma \text{ appears in } A.l \\ & \left.\begin{array}{l} []\ \ C_3 \ \rightarrow\ Act_3 \\ []\ \ C_4 \ \rightarrow\ Act_4 \end{array}\right\} & \text{if } \sigma \text{ appears in } A.r \\ & \textbf{fi} & \end{array}$$

The values of C_i and Act_i are computed as follows,

$$\begin{array}{lcl} C_1 & = & (\textbf{and}\ (\textbf{eq}\ \mathsf{choiceState}\ \mathsf{none}) \\ & & \quad \mathcal{S}_{init}(A.l,\ Cond(\tau_{event}(\sigma, A.l)))) \end{array}$$

$$C_2 \quad = \quad (\textbf{and}\ (\textbf{eq}\ \mathsf{choiceState}\ \mathsf{left})\ Cond(\tau_{event}(\sigma, A.l)))$$

$$\begin{array}{lcl} C_3 & = & (\textbf{and}\ (\textbf{eq}\ \mathsf{choiceState}\ \mathsf{none}) \\ & & \quad \mathcal{S}_{init}(A.r,\ Cond(\tau_{event}(\sigma, A.r)))) \end{array}$$

$$\begin{array}{lcl} C_4 & = & (\textbf{and}\ (\textbf{eq}\ \mathsf{choiceState}\ \mathsf{right})\ Cond(\tau_{event}(\sigma, A.r))) \\ Act_1 & = & \mathsf{choiceState} := \mathsf{left};\ \mathcal{S}_{init}(A.l,\ \langle\tau_{event}(\sigma, A.l)\rangle);\ A.Act_{astd} \\ Act_2 & = & \tau_{event}(\sigma, A.l);\ A.Act_{astd} \\ Act_3 & = & \mathsf{choiceState} := \mathsf{right};\ \mathcal{S}_{init}(A.l,\ \langle\tau_{event}(\sigma, A.r)\rangle);\ A.Act_{astd} \\ Act_4 & = & \tau_{event}(\sigma, A.r);\ A.Act_{astd} \end{array}$$

The conditions C_1 means that no choice has been made and the initial state of left side can execute *sigma*. Act_1 initializes the left component before executing it. The condition C_2 means that the left side has been already chosen and it executes the subsequent action Act_2. A symmetric behavior is generated for the right side.

QChoice ASTD. Let A be a quantified choice ASTD [310] that has the following structure:

$$\langle |\ :, x, T, b \rangle$$

where $x \in \mathsf{Var}$ is the quantification variable, T is a type and $b \in \mathsf{ASTD}$ is the quantified ASTD. The state of A is a structure $\langle |\ :_\circ, [\bot \mid c], E, [\bot \mid s] \rangle$ where $|\ :_\circ$ is the constructor of the quantified choice state, $\bot$ is a constant indicating that the choice has not been made yet, $c \in \mathsf{Term}$ denotes the current value of the choice quantified variable once the choice has been made, E the values of attributes and $s \in \mathsf{States}$.

State and Attributes. The types generated for quantified choice A are defined as follows:

$$\begin{aligned}
\tau_{type}(A) \triangleq\ & \textbf{def type } \mathsf{TState_\$}A \ = \ \textbf{record } \{ \\
& [\![,\ \$v.name : \$v.T]\!]_{v\ \in\ A.Attr} \\
& x : A.x.T, \\
& \mathsf{qchoiceState} : A.x.T, \\
& \mathsf{ts_\$}b : \mathsf{TState_\$}b \\
& \}; \\
& \textbf{const } \mathsf{T_\$}A : \mathsf{Set}\langle A.x.T \rangle = A.T.values; \\
& \tau_{type}(A.b)
\end{aligned}$$

where $\mathsf{TState\_\$}A$ holds the attributes of A and the state of the body $A.b$. The state variable $\mathsf{qchoiceState}$, of type $A.x.T$, stores the current state of A. $\mathsf{T\_\$}A$ is a constant set of type $\mathsf{Set}\langle A.x.T \rangle$ with value $A.T.values$. $\tau_{type}(A.b)$ generates types and variables of $A.b$. For a quantified choice ASTD, $\iota(A)$ and $\phi(A)$ are computed as follows,

$$\iota(A) = \mathsf{qchoiceState} \ := \ \mathbf{nil};$$
$$[\![;\ \$v.name \ := \ \$v.init]\!]_{v \ \in \ A.Attr}$$
$$\phi(A) = (\mathbf{or}\ (\mathbf{and}\ (\mathbf{eq}\ \mathsf{qchoiceState}\ \mathbf{nil})\ \mathsf{exists}(x,\ \mathsf{T\_}\$A,\ \mathcal{S}_{init}(A.b, \phi(A.b))))$$
$$(\mathbf{and}\ (\mathbf{neq}\ \mathsf{qchoiceState}\ \mathbf{nil})\ [x \ := \ \mathsf{qchoiceState}]\ \phi(A.b)))$$

The operation $\mathsf{exists}(c, T, C)$ finds if there exists a value $c \in T$ that can satisfy the condition C, and sets x to that value. The variable qchoiceState has an initial value **nil**; it represents the case when a choice has not been made yet. It takes value $c \in \mathsf{T\_}\$A$ when a choice is made.

Event. Let σ be an event that appears in $A.b$. For a quantified choice ASTD where $\sigma \in \Sigma_A$, the translation of the body of σ is computed as follows,

$$\tau_{event}(\sigma, A) \triangleq$$
$$\mathbf{var}\ \mathsf{c}\ :\ A.x.T;$$
$$\mathbf{if}\ (\mathbf{and}\ (\mathbf{eq}\ \mathsf{qchoiceState}\ \mathbf{nil})$$
$$\mathsf{exists}(\mathsf{c},\ \mathsf{T\_}\$A,\ \mathcal{S}_{init}(A.b, Cond(\tau_{event}(\sigma, A.b))))) \rightarrow$$
$$\mathsf{qchoiceState}\ :=\ \mathsf{c};\ \mathcal{S}_{init}(A.b,\ \langle \tau_{event}(\sigma, A.b)\rangle);\ A.Act_{astd}$$
$$[]\ (\mathbf{neq}\ \mathsf{qchoiceState}\ \mathbf{nil})\ \rightarrow\ \tau_{event}(\sigma, A.b);\ A.Act_{astd}$$
$$\mathbf{fi}$$

When $\sigma \notin \Sigma_A$, $\tau_{event}(\sigma, A) = \mathbf{skip}$. If the operation exists holds, the current state of A is updated to c and the body $A.b$ is initialized and executed. Next, if a choice has been already made (i.e., (**neq** qchoiceState **nil**)), the body $A.b$ is executed.

Call ASTD. Let A be a call ASTD [310]. Recall that A has structure $\langle \mathsf{cal}, n(\vec{c})\rangle$, where n is the name of an ASTD $q = \langle n, P, V, A_{astd}\rangle$. Let $P = \vec{x} : \vec{T}$. For each $c_i \in \vec{c}$, we have $c_i \in T_i$. The type of an ASTD call state is $\langle \mathsf{cal}_\circ, E, [\bot \mid s]\rangle$, where

$\mathsf{cal}_\circ$ is the constructor of the call state, E the values of attributes, $\perp$ denotes that the call has not been made yet and $s \in \mathsf{States}$ is the state of the called ASTD q once the called has been made.

State and Attributes. The types generated for call A are defined as follows:

$$
\begin{aligned}
\tau_{type}(A) \triangleq\ & \textbf{def type } \mathsf{TState\_}\$A \ = \ \textbf{record } \{ \\
& [\![\ \mathsf{,} \ \$v.name : \$v.T]\!]_{v \ \in \ A.Attr} \\
& \mathsf{callState} : \mathsf{CallState}, \\
& \mathsf{ts\_}\$q : \mathsf{TState\_}\$q \\
& \}; \\
& \tau_{type}(A.q)
\end{aligned}
$$

where $\mathsf{TState\_}\$A$ holds the attributes of A and the state of the called ASTD $A.q$. The state variable $\mathsf{callState}$, of type $\mathsf{CallState} = \{\mathsf{called}, \ \mathsf{notcalled}\}$, stores the current state of A. $\tau_{type}(A.q)$ generates types and variables from $A.q$. For a call ASTD, $\iota(A)$ and $\phi(A)$ are computed as follows,

$$
\begin{aligned}
\iota(A) = {} & \mathsf{callState} \ := \ \mathsf{notcalled}; \\
& [\![\ \mathsf{;} \ \$v.name \ := \ \$v.init]\!]_{v \ \in \ A.Attr} \\
\phi(A) = {} & (\textbf{or } (\textbf{and } (\textbf{eq } \mathsf{callState\ notcalled}) \ [\ \$x.name \ := \ \$c.name \ ]_{(x, \ c) \ \in \ x:\vec{c}} \ \mathcal{S}_{init}(A.q, \phi(A.q))) \\
& (\textbf{and } (\textbf{eq } \mathsf{callState\ called}) \ [\![\ \$x.name \ := \ \$c.name \ ]\!]_{(x, \ c) \ \in \ \vec{x}:\vec{c}} \ \phi(A.q)))
\end{aligned}
$$

After calling $\iota(A)$, the variable $\mathsf{callState}$ has an initial value $\mathsf{notcalled}$; it represents the case when A has not been called yet. It takes value called when A is called with input values $\vec{c}$. We use the values $\vec{c}$ of the call to replace the value of parameters $\vec{x}$ in the body $A.q$.

Event. Let σ an event that can appear in $A.q$. For a call ASTD, the translation of the body of σ is computed as follows,

$\tau_{event}(\sigma, A) \triangleq$

if (**and** (**eq** callState notcalled) $\mathcal{S}_{init}(A.q, Cond(\tau_{event}(\sigma, A.q)))) \rightarrow$
  callState := called; $[\vec{x} := \vec{c}]$ $\mathcal{S}_{init}(A.q, \langle\tau_{event}(\sigma, A.q)\rangle)$; $A.Act_{astd}$
[] (**eq** callState called) $\rightarrow$ $[\vec{x} := \vec{c}]$ $\tau_{event}(\sigma, A.q)$; $A.Act_{astd}$
fi

The first condition states that the call has not started (i.e., (**eq** callState notcalled)) and the precondition $\mathcal{S}_{init}(A.q, \tau_{event}(\sigma, A.q))$ must hold. If it is the case, the state of A is updated to called, the body $A.q$ is initialized and executed. The second condition states that if A has already started its execution, it just continues to executes the body $A.q$.

Guard ASTD. Let A be a guard ASTD [310] with the following structure:

$$\text{Guard} \triangleq \langle \Rightarrow, g, b \rangle$$

where $b \in$ ASTD is the body of the guard. The type of a guard state is $\langle \Rightarrow_\circ, E, started?, s \rangle$ where *started*? denotes when the guard has been satisfied, $s \in$ States and E the attribute values of the guard ASTD.

State and Attributes. The types generated for guard A are defined as follows:

$\tau_{type}(A) \triangleq$ **def type** TState_A = **record {**
  $[\![$, $v.name : v.T]\!]_{v \in A.Attr}$
  guardState : GuardState,
  ts_b : TState_b
};
$\tau_{type}(A.b)$

where TState_A holds the attributes of A and the state of the body $A.b$. The state

variable guardState is of type GuardState = {started, notstarted}. $\tau_{type}(A.b)$ generates types and variables from $A.b$. For a guard ASTD, $\iota(A)$ and $\phi(A)$ are computed as follows,

$$\iota(A) = \text{guardState} := \text{notstarted};$$
$$[\![; \$v.name := \$v.init]\!]_{v \in A.Attr}$$
$$\iota(A.b)$$
$$\phi(A) = (\textbf{or } (\textbf{and } (\textbf{eq guardState notstarted}) (\textbf{and } A.g\ \mathcal{S}_{init}(A.b, \phi(A.b))))$$
$$(\textbf{and } (\textbf{eq guardState started})\ \phi(A.b)))$$

After calling $\iota(A)$, the variable guardState has an initial value notstarted; it represents the case when the guard ASTD has not started yet. It takes value started when the guard is satisfied, and execution has started.

Event. Let σ be an event that appears in $A.b$. For a call ASTD, the translation of the body of σ is computed as follows:

$$\tau_{event}(\sigma, A) \triangleq$$
$$\textbf{if } (\textbf{and } (\textbf{eq guardState notstarted})$$
$$(\textbf{and } A.g\ \mathcal{S}_{init}(A.b, Cond(\tau_{event}(\sigma, A.b))))) \rightarrow$$
$$\text{guardState} := \text{started};\ \mathcal{S}_{init}(A.b,\ \langle \tau_{event}(\sigma, A.b) \rangle);\ A.Act_{astd}$$
$$[]\ (\textbf{eq guardState started}) \rightarrow \tau_{event}(\sigma, A.b);\ A.Act_{astd}$$
$$\textbf{fi}$$

The first condition states that the guard has not started (i.e., (**eq** guardState notstarted)), the guard g and the precondition $\mathcal{S}_{init}(A.b, \tau_{event}(\sigma, A.b))$ must hold. If it is the case, the state of A is updated to started, the body $A.b$ is initialized and executed. The second condition states that if A has already started its execution, it just continues to executes the body $A.b$.

Table 4.2 – Example of translation from IL to C++ and Java

IL	**C++**	**Java**
(eq A B)	A.compare(B) == 0	A.compareTo(B) == 0
(or A B)	A || B	A || B
(and A B)	A && B	A && B
(not A)	! A	! A
(in A B)	std::find(B.begin(), B.end(), A) !=B.end()	B.contains(A)
if A1 ->B1 [] A2 ->B2 fi	if (A1) {B1} else if (A2) {B2}	if (A1) {B1} else if (A2) {B2}
while(A) begin B end	while(A) { B }	while(A) { B }
for(in A B) C end	for(const T& A : B) { C }, *where T is type of A*	for(T A : B) { C }, *where T is type of A*
exists(x, T, A)	bool exists(x){A}, *where domain T declared globally*	boolean exists(x){A} *where domain T declared globally*
forall(x, T, A)	bool forall(x){A}, *where domain T declared globally*	boolean forall(x){A} *where domain T declared globally*
var x : type = value	type x = value	type x = value
main(P)	int main(P)	public static void main(P)
e = read_event(src)	Event<T> e = read_event(src)	Event<T> e = read_event(src)

4.3.2 Translation from the Intermediate Model to the target code

The translation operation takes an input IL model *ilm*, a given target language *lang* (e.g., C++), and returns the corresponding source code *code* in this language. The translation approach constructs the source code by traversing types, variables, functions and statements in the IL model. The main procedure for translation is expressed as follows,

Algorithm 1 Generic structure of the translation

1: **procedure** Trans (ilm, $lang$, $code$)
2: Trans$_t$ ($ilm.typedecls$, $lang$, $code$)
3: Trans$_d$ ($ilm.vardecls$, $lang$, $code$)
4: Trans$_f$ ($ilm.functions$, $lang$, $code$)
5: **end procedure**

The procedure Trans generates the source code from the declaration of types $ilm.typedecls$, declaration of variables $ilm.vardecls$ and $ilm.functions$ (see Fig. 4.3). The procedure Trans$_t$ translates the declaration of types in the target language. Likewise, the procedure Trans$_d$ translates variables in the target language. The procedure Trans$_f$ translates event functions and the main function in the target language. Wrapping and template types are used to deal with event variants and complex event

Table 4.3 – Example of translation of types

IL	**C++**	**Java**
int	int	int
Boolean	bool	boolean
string	std::string	String
Set<T>	std::set<T>	Set<T>
Map<T, U>	std::map<T, U>	Map<T, U>
Json	nlohmann::json	JSONObject
def type A = record { B };	struct A { B };	class A { B }
def type A = enum { B };	enum A { B };	enum A { B }
TState_A $\oplus TState\_B$	TState; it is the super class of TState_A and TState_B	TState; it is the super class of TState_A and TState_B

types like JSON (e.g., string, JSONObject). JSON types are used to represent complex event records (e.g., logs). For example, Table 4.2 shows the translation relation between the IL language and the C++/Java languages.

Table 4.3 also shows the translation of types between the IL language and C++/Java languages. The C++ type `json` is based on the Nlohmann library and it is used to represent complex records. The library allows to easily manipulate JSON documents.

4.3.3 Optimization

During code generation, we apply three levels of optimization: **if-fi** optimization, kappa optimization, and native-compiler optimization.

a) ***if-fi*** *optimization.* It consists of computing **if-fi** conditions only once in the generated code; thus it improves the execution time. To implement this approach, we add Boolean variables $cond_i$ in the type of the ASTD state to store the values of the conditions. For example, the code of the type of A is expressed in C++ as follows,

$$\begin{aligned}\tau_{type}(A) \triangleq struct\ \mathsf{TState\_}A\ =\ \{\\ &\mathsf{TState\_}B\ \mathsf{ts\_}B;\\ &\mathsf{TState\_}E\ \mathsf{ts\_}E;\\ &bool\ \mathsf{cond}_1;\\ &bool\ \mathsf{cond}_2;\\ &bool\ \mathsf{cond}_3;\\ \};\end{aligned}$$

The event functions are also modified to compute conditions only once. Let assume the event function a in the simplified form,

```
...
void a(int u)
{
    ...
    if(C1 || C2)
    {
         if(C1)
         {
           A1
         }
         else if (C2)
         {
           A2
         }
     }
}
...
```

The previous code after **if-fi** optimization is expressed as follows,

```
...
```

```
void a(int u)
{
    ...
    cond_1 = ((cond_2 = C1) || (cond_3 = C2));
    if(cond_1)
    {
        if(cond_2)
        {
          A1
        }
        else if(cond_3)
        {
          A2
        }
    }
    ...
}
...
```

b) κ-optimization. It allows to reduce the complexity of exists and forall operations in large quantified ASTDs [94, 95, 279]. The algorithm prunes the search space in the quantification domain using the arguments of an event. Then, only the correct copy of an ASTD that can execute the current event is executed. This operation is done in logarithmic time.

c) native-compiler optimization. After **if**-**fi** and kappa optimization, the generated code is optimized for code size, compilation and execution time using native compiler options (e.g. -O*level* in GCC, -XX:*Options* in JIT). The options -O*level* includes flags -fcompare-elim, -fdce, -ftree-dce, -fmove-loop-invariants that allows to remove dead codes and loop invariants.

4.4 Tool support

We have designed and developed cASTD [2] in Java following the Continuous integration (CI) and continuous delivery (CD) approach. We have chosen Java because of its portability. The Java Virtual Machine (JVM) allows cASTD to be executed on different operating systems (OS). It also supports a Just-In-Time (JIT) compiler that optimizes the compilation of bytecode into machine code. The current version of cASTD generates efficient code in C++ thanks to the IL language.

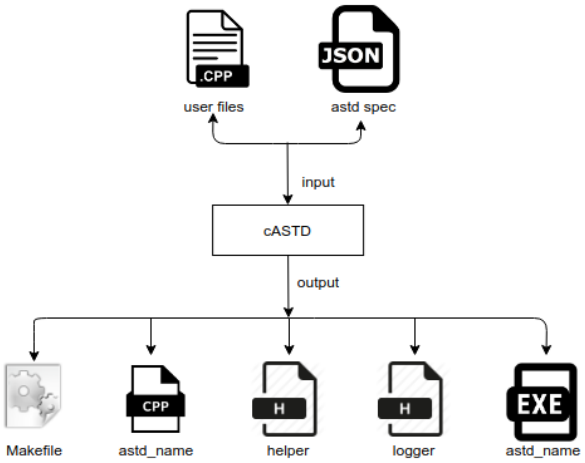


Figure 4.5 – cASTD - Structure of input/output files

cASTD is executed in command-line. It takes three inputs : the input user files that contains action codes, the ASTD specification, and the target language. cASTD has several options including *-c / –cond-opt* for **if-fi** optimization, *-d [false / true]* for debugging and logging executing actions, and *-k [direct / indirect]* for kappa optimization. In Fig. 4.5, cASTD generates the source code in the target language and the associated program (monitor) that will be executed on traces. The source code consists of the *helper* file that calls constructors associated to given string types, the *logger* file that allows one to debug the generated program (only in debug mode), the *astd_name* file that contains the translated code of the ASTD specifications, and the *makefile* for linking and compilation. This makefile calls the native compiler

2. The compiler is available at https://depot.gril.usherbrooke.ca/lionel-tidjon/castd

corresponding the target language (e.g. GCC for C++, JIT for Java) and it is automatically executed by cASTD to produce the *astd_name* program. Configurations about the native compiler is provided in a YAML file. It includes the name, the class path, and compilation arguments. When external static/dynamic libraries are used by action codes in ASTD specifications, they must be also referenced in the YAML file for compilation.

4.5 Related work

S. L. M. Barrocas et al. [46] proposes the JCircus compiler that translates specifications written in Circus (a combination of Z and CSP) into Java programs that use JCSP, a library that implements most of the CSP constructs. Baringer et *al.* [30] introduced the concept of Quantified Event Automata (QEA) that applies quantified variables on event automata. QEA allows efficient execution by slicing parametric traces into a set of propositional traces that can be processed by separate propositional monitors. QEA is implemented in a tool called MARQ. ASTD supports a richer set of operators than JCircus and MARQ which can be arbitrarily combined in an algebraic manner.

In big data, Complex Event Processing (CEP) supports several processors including filtering, slicing, windowing, and piping that allows one to query and process multiple event streams. Beepbeep [320] combines CEP with logic operators from Linear Temporal Logic (LTL), Finite State Machines (FSMs), and First-order quantifiers like QEAs. CEP processors can be expressed in the ASTD language in a modular and concise manner. ASTDs can be seen as extensions of FSMs.

Larva [68] allows one to specify properties of Java programs using variables, events, states, and transitions. The Larva tool compiles these specifications (scripts) into Java byte code which is later executed on event traces in real-time. Larva does not support composition operators like ASTDs; it essentially relies on basic automatons.

MonPoly [35] processes a stream of system events with identifiers representing the data involved and reports policy violations. Policies are given as formulas of an expressive safety fragment of metric first-order temporal logic (MFOTL). MUFIN [83] is a tool based on AspectJ that uses union-find structure for monitoring the individual

behaviour and interaction of an unbounded number of runtime objects. Union-find structure is based on projection automata [83] and it represents disjoint sets of objects organised as a tree. ASTD automata can be seen as extensions of projection automata.

4.6 Performance evaluation

In this section, we propose to evaluate the performance of cASTD against iASTD and other event processing tools in the literature using some case-studies.

4.6.1 Case studies

The runtime verification competition 2016 (RVC2016) consisted of two tracks : Offline and JAVA. The offline tracks consisted of 3 benchmarks from MARQ [30, 261] (i.e., *AunctionBidding*, *CandidateSelection*, and *SQLInjection*) and 3 benchmarks from BeepBeep v3 [320] (i.e., *PinguCreation*, *EndlessBashing*, and *TurnAround*). The JAVA tracks contains 9 benchmarks : 3 benchmarks from LARVA [68] (i.e., *GreyListing*, *ReconcileAccounts*, and *Logging*), 3 benchmarks from MARQ [30, 261] (i.e., *PublisherSubscriber*, *AnnonyingFriend*, and *ResourceLifecycle*), and 3 benchmarks from MUFIN [83] (i.e., *Tree*, *Multiplexer*, *Toggle*). In the offline track, we have selected 3 benchmarks from MARQ. We have also selected all benchmarks from the JAVA track. Since each benchmark contained few event samples, we have generated 1 million events for each benchmark using existing event samples.

We have also considered 8 other scenarios. Seven scenarios consist of complex ASTD specifications generated from the eASTD editor. Each of them contains one hundred top ASTDs combined with operators such as Flow, QInterleaving, QChoice, QSynchronization, Sequence, Kleene, and Automaton. The last scenario is called HasNext. It consists of two cases: (1) A file of one million of events containing hasNext and Next patterns, and (2) one million of events randomly generated.

4.6.2 Generated Code Efficiency

In this section, we compare the code generated from cASTD to iASTD and other event processing tools in the literature. The comparison is done in terms of execution

time on a single computer (chip type: NVIDIA GeForce GTX 1050, CPU: 2.80 GHz-8 cores, RAM: 16GB). The performance of cASTD and iASTD have been evaluated on the RVC 2016 benchmarks and complex scenarios generated from the eASTD editor. The scenarios are available on Git [3]. cASTD and iASTD were compared to other tools such as Beepbeep v1, Beepbeep v3 and MonPoly [35] using the LABPAL framework [138]. The LABPAL framework allows one to easily run several experiments and it generates reports after execution. We were unable to replay MUFIN, LARVA, and MARQ specifications using the RVC 2016 benchmark; because they were not available, and their usage specification was not sufficiently documented.

4.6.2.1 Comparison with iASTD

In Table 4.4, we present execution time of cASTD and iASTD. We have processed 1 million of events using 7 specifications generated by the eASTD editor (see Table 4.4) and 3 specifications from each benchmark (see Table 4.5). As result, cASTD is about 10x more efficient than iASTD. The specifications that use the flow operator are faster than those based on the sequence operator. In addition, the high number of enclosing quantifications does not affect the performance of iASTD and cASTD, thanks to kappa optimization.

Table 4.4 – Comparison between cASTD and iASTD using generated specifications

Scenario	Execution time	
	cASTD(s)	iASTD(s)
flow+aut (x100)	0.425	4.495
seq+aut (x100)	0.550	5.738
flow+kleene+aut (x100)	0.370	3.109
flow+qinter+aut (x100)	0.448	3.329
flow+qchoice+aut (x100)	0.391	3.395
flow+qinter+qchoice+aut (x100)	0.304	3.134
flow+qsynch+qinter+qchoice+kleene+aut (x100)	0.289	2.940

In Table 4.5, we compare cASTD and iASTD on benchmarks (LARVA, MUFIN,

3. https://depot.gril.usherbrooke.ca/lionel-tidjon/castd/tree/master/tests

MARQ) from the runtime verification competition 2016. Each benchmark is numbered from 1 to 3. For MARQ, we have Bench 1 = *AunctionBidding*, Bench 2 = *CandidateSelection*, and Bench 3 = *SQLInjection*. Overall, cASTD provides a 10x performance improvement over iASTD.

Table 4.5 – Comparison between cASTD and iASTD using RVC 2016

Scenario	LARVA		MUFIN		MARQ	
	cASTD(s)	iASTD(s)	cASTD(s)	iASTD(s)	cASTD(s)	iASTD(s)
Bench 1	0.044	0.384	0.030	0.320	0.075	0.233
Bench 2	0.081	0.422	0.061	0.408	0.098	0.249
Bench 3	0.022	0.299	0.059	0.317	0.084	0.239

4.6.2.2 Comparison with other tools

We have executed MonPoly, iASTD, cASTD, Beepbeep v1 and Beepbeep v3 on the HasNext scenario using the LABPAL framework. In Table 4.6, cASTD and iASTD achieved better performance than Beepbeep v1, Beepbeep v3, and MonPoly. Beepbeep v1 is faster than Beepbeep v3 on the HasNext scenario. cASTD provides the best performance of all tools.

Table 4.6 – Comparison between Beepbeep v1, Beepbeep v3, MonPoly, iASTD, and cASTD (milliseconds)

Scenario	**Beepbeep v3**	**Beepbeep v1**	**MonPoly**	**iASTD**	**cASTD**
HasNext(file)	143061.52	300.58282	0.829875	2.551020	0.29585
HasNext(gen)	555555.56	7396.4497	113636.37	18148.82	3371.544

4.7 Conclusion

In this work, we provided translation rules to efficiently generate code from ASTD specifications using an intermediate language which can be easily translated into conventional programming languages like C++ and Java. Our approach is implemented in a tool called cASTD whose performance is about 10x faster than iASTD, and it is also faster than other event processing tools (Beepbeep v3, iASTD, MonPoly). In large and complex specifications, cASTD achieved better results than other tools thanks to several optimization mechanisms such as **if-fi** optimization, kappa optimization, and native optimization. In the future, we intent to evaluate cASTD on more complex real-world cases in cybersecurity and information systems.

Conclusion

Plusieurs outils tels que Snort et Zeek permettent de spécifier des comportements d'attaques en utilisant des règles. Toutefois, ces règles deviennent inefficaces face à certaines attaques complexes et difficiles à maintenir lorsque le nombre d'attaques est potentiellement grand. De ce fait, cette thèse a proposé un langage *avec état*, graphique, modulaire et exécutable afin de corréler plusieurs sources d'événements et d'identifier des attaques simples et complexes. Dans cette optique, il était question d'analyser les comportements d'attaques et de tenter de les spécifier avec le langage ASTD existant afin d'identifier ses limites. Ces limites ont permis d'ajouter des fonctionnalités aux ASTDs et de définir une sémantique formelle pour la notation étendue [234]. Ensuite, la notation étendue a été testée avec des bases d'attaques telles que CAPEC et ATT&CK afin d'offrir une vue holistique des phases et des étapes de l'attaque [312].

Plusieurs études de cas d'attaques provenant de Nokia Canada et d'ensembles de données de référence ont été spécifiées avec les ASTDs [312]. Ces spécifications d'attaques ont été exécutées à l'aide d'un interpréteur d'ASTD dont les performances ont été comparées avec des outils industriels de détection d'attaques tels que Snort et Zeek. L'interpréteur produit peu de fausses alertes et est plus précis. Toutefois, le temps d'exécution de l'interpréteur est relativement élevé par rapport à celui de Snort et Zeek. De ce fait, la prochaine étape a consisté à réaliser un compilateur d'ASTDs vers plusieurs langages de programmation tels que C++ et Java, en utilisant un langage intermédiaire, afin d'améliorer le temps d'exécution. Pour ce faire, il était question de formaliser les règles de traduction du langage ASTD vers le langage intermédiaire et du langage intermédiaire vers les langages de programmation. Ensuite, le code généré, optimisé et compilé a été comparé avec plusieurs outils de

traitement de flux d'événements (iASTD, Beepbeep, MonPoly) sur des ensembles de données de référence. Lors de l'exécution, les programmes compilés sont plus rapides que l'interpréteur iASTD et d'autres outils (i.e., Beepbeep, MonPoly).

Synthèse des contributions

La première contribution de cette thèse a consisté en la définition formelle de la syntaxe et de la sémantique d'une notation adaptée pour la spécification des attaques récentes en utilisant les ASTDs [234]. Le langage est supporté par un interpréteur, qui permet de corréler des d'événements (paquets, logs).

Afin de représenter de façon concise les phases de l'attaque, les ASTDs ont été couplés avec les bases de scénarios d'attaques tels que CAPEC et ATT&CK. L'approche a été validée sur plusieurs attaques du monde réel et des bases d'attaques existantes répertoriées dans la littérature [312]. L'interpréteur a fourni un taux de précision élevé pour le traitement de diverses sources d'événements par rapport à Snort et Zeek, ce qui est un facteur important pour la détection d'intrusions.

Le temps d'exécution de l'interpréteur des ASTDs était élevé par rapport à Zeek et Snort face à des flots d'événements importants. De ce fait, un compilateur a été développé afin de générer du code exécutable efficient à partir des spécifications ASTDs. Lors de la validation de l'outil, les programmes compilés étaient plus efficaces que les outils existants, grâce à plusieurs optimisations du code telles que l'élimination des calculs redondants et l'optimisation Kappa.

Menaces à la validité

Plusieurs difficultés ont été rencontrées lors de la validation des spécifications d'attaques et des outils implémentés tout au long de cette thèse. En ce qui concerne la validation des spécifications, les menaces suivantes ont été identifiées :

— Dans le cas des attaques complexes, l'écriture de certaines spécifications nécessitait une analyse de milliers voire de millions d'événements, pouvant affecter la prise en compte de tous les scénarios de l'attaque ;

— 45% des spécifications d'attaques ont été vérifiées avec des experts provenant de Nokia Canada et du Centre de la sécurité des télécommunications. 55% des spécifications ont été validées par nous-même, pouvant ainsi biaiser la validité des spécifications.

Durant la validation des outils implémentés dans le cadre de cette thèse, plusieurs menaces ont été rencontrées parmi lesquelles,

— L'évaluation des outils a été effectuée à 50% avec des ensembles de données de référence, 20% avec les données du monde réel dans les environnements industriels tels que Nokia, et 30% avec les données simulées par nous-même sur des bancs d'essai, sujettes à plusieurs erreurs de manipulation ;

— La comparaison des outils était très coûteuse en temps car il fallait exécuter plusieurs fois les outils sur des millions d'événements afin de s'assurer de l'intégrité des résultats. Plusieurs erreurs d'expérimentation étaient susceptibles de survenir durant ces opérations manuelles.

Perspectives

La suite de ce travail pourrait consister au passage à l'échelle des outils implémentés, l'implémentation d'un vérificateur de modèles ASTDs, l'ajout de nouvelles fonctionnalités aux ASTDs pour l'exécution dans des environnements distribués, et la vérification formelle du code dans le langage intermédiaire avant de le compiler en exécutable. Une fois que le passage à l'échelle est effectué, les outils pourront être utilisés dans de larges infrastructures pour la détection des attaques. De plus, le vérificateur de modèles ASTDs permettra de s'assurer que les spécifications sont bien écrites et que le code des actions est valide avant leur exécution. Actuellement, les spécifications ne peuvent pas être exécutées sur plusieurs environnements en parallèle (ex. réseau, hôte). La sémantique formelle doit être modifiée afin de prendre en compte ces changements. Enfin, le code produit dans la représentation intermédiaire doit être conforme à la spécification ASTD d'origine. De ce fait, une vérification formelle serait souhaitable avant de générer le code dans un langage de programmation.

Annexe A

Détection d'intrusion à base des ASTDs

A.1 Description détaillée de Gancrab

Dans ce qui suit, une variante de rançongiciels appelée Gancrab est spécifiée. Gandcrab est un rançongiciel mis à jour et actif qui est publié sur plusieurs versions (i.e., version 1.0 à 5.2). Chaque variante vise à crypter les fichiers utilisateurs avec les extensions .GDCB, .CRAB et .KRAB. Ils déposent également une note de rançon contenant les instructions pour le paiement sur les domaines Tor (par exemple gandcrabmfe6mnef.onion, gandcrab2pie73et.onion). La description détaillée des actions de Gancrab est présentée ci-dessous.

1. L'attaquant délivre un courriel contenant un fichier incorporé avec les extensions .DOC, .PDF, .ZIP et .VBS. Le fichier intégré semble inoffensif et contient un lien malveillant. Un extrait du courriel envoyé à l'aide de l'outil Social Engineering Toolkit (SET) est de la forme,

```
220 smtp.gmail.com ESMTP w6sm3177398iom.22 - gsmtp
ehlo ip-172-31-22-90.us-east-2.compute.internal
250-smtp.gmail.com at your service, [18.216.143.79]
250-SIZE 35882577
250-8BITMIME
250-STARTTLS
250-ENHANCEDSTATUSCODES
250-PIPELINING
250-CHUNKING
250 SMTPUTF8
STARTTLS
220 2.0.0 Ready to start TLS
............R..Nj. .....q.%.1[....E.0h....L....0.,.2.../.+.1.-..(.$..
```

2. Une fois que la victime clique et exécute le fichier, elle télécharge Gandcrab ("kukul.exe") et l'exécute. Cette opération est observée dans le trafic HTTP ci-dessous. Après cela, le fichier se ferme mais fonctionne toujours en arrière-plan.

```
GET /js/kukul.exe HTTP/1.1
Connection: Keep-Alive
Accept: */*
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64;
Trident/7.0; rv:11.0) like Gecko
Host: 172.104.40.92
```

3. Gandcrab obtient l'adresse IP de la victime en envoyant une requête DNS au site Web ipv4bot.whatismyipaddress.com. Ce comportement est observé dans le trafic DNS ci-dessous.

```
00000  36 27 01 00 00 01 00 00  00 00 00 00 07 69 70 76   6'...... .....ipv
00010  34 62 6f 74 11 77 68 61  74 69 73 6d 79 69 70 61   4bot.wha tismyipa
00020  64 64 72 65 73 73 03 63  6f 6d 00 00 01 00 01      ddress.c om.....
```

Le crypto-ver obtient également d'autres informations système (par exemple, version du système d'exploitation, nom d'utilisateur, domaine Active Directory, architecture du processeur, numéro de série du volume de stockage principal, antivirus en cours d'exécution).

4. Gandcrab se réplique en créant un fichier malveillant dans le dossier AppData (par exemple, yxvace.exe). Ce fichier malveillant vérifie plusieurs sites de commande et contrôle (C2) à l'aide de la commande nslookup site_name dns_server. Ces sites C2 incluent carder.bit, ransomware.bit et zonealarm.bit. Les serveurs DNS incluent ns1.wowservers.ru et ns2.wowservers.ru. Ce comportement peut être observé dans les

journaux d'événements Windows comme suit,

```
Provider_Name: Microsoft-Windows-Sysmon
Event_ID: Process Create
Computer: WIN-N58C5B48R34
Data_ProcessID: 69104
Data_Image: C:\Windows\SysWOW64\nslookup.exe
Data_CommandLine: nslookup carder.bit ns2.wowservers.ru
Data_CurrentDirectory: C:\Windows\system32\
Data_User: WIN-N58C5B48R34\Admin
Data_ParentProcessId: 1132
Data_ParentCommandLine: C:\Users\Admin\AppData\Roaming\Microsoft\yxvace.exe
```

5. Lors de la vérification du serveur C2, Gandcrab envoie également une requête HTTP GET à celui-ci,

```
GET / HTTP/1.1
Host: carder.bit
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64;
Trident/7.0; rv:11.0) like Gecko
Cache-Control: no-cache
```

6. Ensuite, Gandcrab crypte les données collectées dans l'action 3 et les envoient sur le serveur C2. Le serveur C2 répond et le cryptage démarre.

```
POST /iafsc?eapeli=ausc&lpee=lpie HTTP/1.1
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64;
Trident/7.0; rv:11.0) like Gecko
Host: carder.bit
Content-Length: 5828
Cache-Control: no-cache

wfKD6iudumBkmpL8IRr4U4KxFFazOXLtyjwmOrT1y1YWvOiWMx5GYaRdvZZTTpZRqHYW
7nxWyLfFTGKHhh5qBJzzs9MC7736UkGSDDniUJJG8/LFF//kmGmoAZAGLo2j5/wd2UrxM
...
```

A.2 Règles Snort

A partir de l'étude de cas précédente, nous pouvons construire 4 règles de détection, chacune faisant référence aux phases CAPEC-98 et CAPEC-549. Une référence CAPEC est ajoutée au système Snort et mappe le mot-clé capec à http ://capec.mitre.org/find/index.html ?q=. Nous utilisons la règle suivante

```
alert tcp $HOME_NET any -> $EXTERNAL_NET 80 (msg:"Net/Win32.Ransom.Gandcrab -
Malicious Software Downloading"; flow:established,from_client; content:"GET";
http_method; content:".exe HTTP/1.";fast_pattern:only; content:"Connection: Keep-Alive";
http_header;content:"Accept|3a 20|"; http_header; content:"User-Agent|3a 20|Mozilla";
http_header; content:"Host|3a 20|"; pcre:"/Host\x3a\x20(:?[0-9]{1,3}\.){3}[0-9]{1,3}/H";
reference:capec,CAPEC-98; classtype:Downloader; sid: 10000000004; rev:1;)
```

pour détecter le téléchargement de Gandcrab via un fichier joint (voir Action 2)). Snort ne peut pas détecter l'action 1 ; puisque le trafic réseau est chiffré. La règle cible le trafic HTTP entrant ($HOME_NET) dirigé vers le serveur C2 ($EXTERNAL _NET). $HOME_NET est une variable de Snort qui contient la plage IP ou le sous-réseau du réseau interne (par exemple $HOME_NET = 10.23.4.0/24, $HOME_NET = [10.10.8.1, 10.10.8.254]). La variable $EXTERNAL_NET définit la plage IP ou le sous-réseau du réseau externe sur lequel le trafic HTTP sortant est activé. La clause any signifie que la règle accepte tous les ports de connexion du trafic entrant. Dans les flux de paquets, il vérifie si la méthode HTTP est GET, l'URI HTTP contient le motif .exe HTTP / 1., La connexion HTTP est maintenue active, l'en-tête Accept est utilisé, l'agent utilisateur est Mozilla et l'en-tête Host est une adresse IP. Le motif .exe HTTP / 1. permet d'identifier le téléchargement du fichier malicieux. La règle suivante

```
alert udp $HOME_NET any -> $EXTERNAL_NET 53 (msg:"Net/Win32.Ransom.Gandcrab -
Got IP address (Spyware)"; content:"|00 01 00 00 0000 00 00|";
content:"ipv4bot|11|whatismyipaddress|03|com"; fast_pattern; reference:capec,
CAPEC-549; classtype:Spyware; threshold:type limit,track by_src,count 1,seconds 60
;sid:10000005; rev:1;)
```

suit le trafic DNS entrant ($HOME_NET) dirigé vers le serveur C2 ($EXTERNAL _NET). Il accepte tous les ports de connexion du trafic entrant. La règle vérifie la requête DNS contient des octets |00 01 00 00 00 00 00 00 | et l'url du site d'espionnage `ipv4bot|11|whatismyipaddress|03|com|`. Le motif ipv4bot|11| whatismyipaddress|03|com permet d'identifier les activités d'espionnage de Gandcrab (voir Action 3). L'option de seuil limite le nombre d'alertes à une par minute. La règle suivante

```
alert tcp $HOME_NET any -> $EXTERNAL_NET 80 (msg:"Net/Win32.Ransom.Gandcrab -
CnC checkin"; flow:established,to_server; urilen:1;content:"GET"; http_method;
content:"/"; http_uri; content:"Host|3a 20|"; http_header;content:".bit";
fast_pattern;content:"User-Agent|3a 20|Mozilla"; http_header; content:
"Cache-Control|3a 20|no-cache"; content:!"Accept"; content:!"Connection";
content:!"Referer"; reference:capec, CAPEC-549; classtype:Ransomware;
sid:10000000006; rev:1;)
```

identifie l'attaquant Action 5. Snort ne peut pas voir l'attaquant Action 4 puisqu'elle est effectuée au niveau de l'hôte. La règle vérifie si la méthode HTTP est GET, l'urilen est 1, l'url de l'hôte provient du domaine .bit, l'agent utilisateur est Mozilla, il n'y a pas de contrôle de cache et il n'y a pas d'en-têtes HTTP tels que Accept,

Connexion et Referer. Le symbole ! désigne la négation (par exemple, le contenu :! "c" ne contient pas "c"). La dernière règle

```
alert tcp $HOME_NET any -> $EXTERNAL_NET 80 (msg:"Net/Win32.Ransom.Gandcrab -
Communicating With Command and Control Server"; flow:established,to_server;
content:"POST"; http_method; content:"Host|3a 20|"; http_header;content:".bit";
fast_pattern; content:"User-Agent|3a 20|Mozilla"; http_header; content:
"Content-Type|3a 20|"; pcre:"/Content-Type\x3a\x20(application|multipart)/";
content:"Cache-Control: no-cache"; http_header; content:!"Accept";
reference:capec,CAPEC-549; classtype:Ransomware;sid:10000000007; rev:1;)
```

détecte le cryptage et l'exfiltration des informations sur la victime (voir Action 6). La règle vérifie si la méthode HTTP est POST, l'url de l'hôte provient du domaine de premier niveau .bit, l'agent utilisateur est Mozilla, le type de contenu est application/x-www-form-urlencoded ou multipart/form-data, il n'y a pas de contrôle de cache et pas d'en-tête Accept.

A.3 Règles Zeek

Dans l'étude de cas, nous avons spécifié les phases de Gandcrab à l'aide de scripts et de règles Zeek [155]. Ci-après, nous décrivons quelques règles Zeek pour la détection de Gancrab. La première règle

```
signature gandcrab_phishing_capec_98 {
 ip-proto == tcp
 dst-port == 80
 payload /^GET \//
 http-request /\.(exe|bin)/
 http-request-header /Connection: Keep-Alive/
 http-request-header /Accept: /
 http-request-header /User-Agent: Mozilla/
 http-request-header /Host: ([0-9]{1,3}\.){3}[0-9]{1,3}/
}
```

détecte le téléchargement de Gandcrab pendant la phase de hameçonnage (voir Action 2). Zeek peut tracer le trafic chiffré pendant l'action 1 de l'attaquant mais il ne peut pas lire son contenu, c'est-à-dire le fichier joint. La prochaine règle

```
signature gandcrab_spyware_capec_549 {
 ip-proto == udp
 dst-port == 53
 payload /\x00\x01\x00\x00\x00\x00\x00\x00/
 payload /ipv4bot\x11whatismyipaddress\x03com/
 requires-signature gandcrab_phishing_capec_98
}
```

identifie le site d'espionnage avec lequel Gandcrab communique (voir Action 3) et nécessite l'exécution de la règle précédente. La règle suivante

```
signature gandcrab_cnc_checkin_capec_549 {
 ip-proto == tcp
 dst-port == 80
 payload /^GET \//
 http-request-header /Host: /
 http-request-header /[a-z]+\.(bit|ru)/
 http-request-header /User-Agent: Mozilla/
 http-request-header /Cache-Control: no-cache/
 requires-signature gandcrab_spyware_capec_549
}
```

détecte quand Gandcrab vérifie ses sites C2 à partir des domaines .bit et .ru (voir Action 5). La dernière règle

```
signature gandcrab_cnc_capec_549 {
 ip-proto == tcp
 dst-port == 80
 payload /^POST \//
 http-request-header /Host: /
 http-request-header /[a-z]+\.(bit|ru)/
 http-request-header /User-Agent: Mozilla/
 http-request-header /Content-Type: application/
 http-request-header /Cache-Control: no-cache/
 requires-signature gandcrab_cnc_checkin_capec_549
 event "Net/Win32.Ransom.Gandcrab - Communicating With CnC Server"
}
```

identifie l'action malveillante 6, c'est-à-dire lorsque Gandcrab envoie les informations de la victime sur le serveur C2. La règle est déclenchée après l'exécution de l'action C2 de Gandcrab.

A.4 Règles OSSEC

OSSEC est un outil IDS qui détecte les activités malveillantes sur un ou plusieurs hôtes. Il utilise des règles de détection, y compris des règles externes de Sysmon, un service système Windows. L'outil a été utilisé lors de la comparaison de Snort, Zeek

et iASTD. Il a été associé à iASTD afin de fournir les événements de l'hôte pour la détection des attaques. Certaines règles OSSEC permettant d'identifier Gancrab sont présentées ci-dessous. La première règle

```
<rule id="1000001" level="9">
  <match>downloads|temp</match>
  <regex>.exe</regex>
  <description>Suspicious File Downloading</description>
  <group>capec,CAPEC-98</group>
</rule>
```

détecte aux fichiers téléchargés suspects avec des extensions .exe (voir Action 2). Ces fichiers sont souvent déposés dans les dossiers de téléchargement et temporaires. La règle suivante

```
<rule id="1000002" level="12">
 <if_sid>1000001</if_sid>
 <match>nslookup</match>
 <regex>.bit|.ru</regex>
 <description>Gandcrab CnC Checkin - DNS lookup</description>
 <group>capec,CAPEC-549</group>
</rule>
```

identifie quand Gandcrab utilise la commande nslookup afin de vérifier ses sites C2 à partir des domaines .bit et .ru (voir Action 4). La règle requiert l'exécution de celle précédente. La règle suivante

```
<rule id="1000003" level="12">
  <if_sid>1000001</if_sid>
  <match>svchost.exe</match>
  <protocol>tcp</protocol>
  <srcport>3389</srcport>
  <description>Gandcrab CnC- Exfiltration</description>
  <group>capec,CAPEC-549</group>
</rule>
```

détecte quand Gandcrab élimine les informations système et se propage sur le port ouvert 3389. La règle ne s'exécute que si un téléchargement de fichier suspect a été effectué. Le dernière règle

```
<rule id="1000004" level="12">
  <if_sid>1000001</if_sid>
  <match>DECRYPT.txt|DECRYPT.html</match>
  <description>Gandcrab CnC - Encrypted Files</description>
  <group>capec,CAPEC-549</group>
</rule>
```

vérifie si une note de déchiffrement a été déposée sur l'hôte. Cela signifie que les fichiers de la victime sont cryptés et qu'il doit payer pour obtenir la clé de décryptage.

Annexe B

Traduction des ASTDs en langages de programmation de haut niveau

Ci-après, nous présentons 3 cas de génération de code (voir Sect. B.1, Sect. B.2, Sect. B.3) et 1 cas d'optimisation de code (voir Sect. B.4).

B.1 Exemple de génération de code : Cas 1

La figure B.1 montre un exemple de flux ASTD contenant deux ASTD Kleene. Chaque ASTD Kleene contient un ASTD automate.

C'est le code C++ généré à partir de la Fig. B.1 par le compilateur ASTD.

```
1  #include "Code.cpp"
2  #include "helper.h"
3  #include "logger.h"
4  enum KleeneState{
5    KLEENE_NOTSTARTED,
6    KLEENE_STARTED
7  };
8  enum AutState{
9    S0,
10   S1
11 };
12 struct TState_C_2{
13   AutState  autState;
14
15 };
```

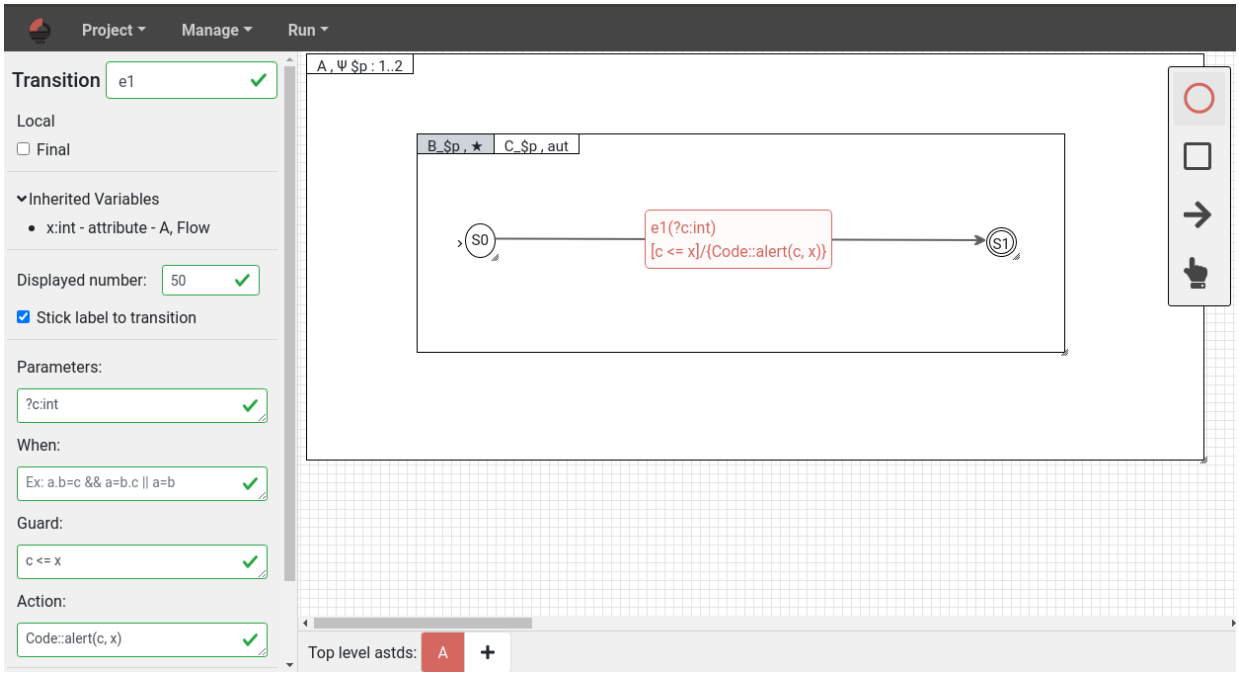


Figure B.1 – Exemple - Flux, Kleene, Automate

```
16  struct TState_B_2{
17    KleeneState  kleeneState;
18    TState_C_2  ts_C_2;
19
20  };
21  struct TState_C_1{
22    AutState  autState;
23
24  };
25  struct TState_B_1{
26    KleeneState  kleeneState;
27    TState_C_1  ts_C_1;
28
29  };
30  struct TState_A{
31    int  x;
32    TState_B_1  ts_B_1;
33    TState_B_2  ts_B_2;
34
35  };
36  const std::vector<AutState>  shallow_final_C_1 = {S1};
37  const std::vector<AutState>  shallow_final_C_2 = {S1};
38  TState_A  ts_A;
39
```

```
40  void e1(int  c){
41          if(((std::find(shallow_final_C_1.begin(), shallow_final_C_1.end(),
42          ts_A.ts_B_1.ts_C_1.autState) !=shallow_final_C_1.end() && (S0 == S0 && c <= ts_A.x))
43          || (ts_A.ts_B_1.ts_C_1.autState == S0 && c <= ts_A.x))){
44                  if((std::find(shallow_final_C_1.begin(), shallow_final_C_1.end(),
45                  ts_A.ts_B_1.ts_C_1.autState) !=shallow_final_C_1.end()
46                  && (S0 == S0 && c <= ts_A.x))){
47                          ts_A.ts_B_1.kleeneState = KLEENE_STARTED;
48                          if((S0 == S0 && c <= ts_A.x)){
49                                  Code::alert(c, ts_A.x);
50                                  ts_A.ts_B_1.ts_C_1.autState = S1;
51
52                          }
53
54                  }else if((ts_A.ts_B_1.ts_C_1.autState == S0 && c <= ts_A.x)){
55                          if((ts_A.ts_B_1.ts_C_1.autState == S0 && c <= ts_A.x)){
56                                  Code::alert(c, ts_A.x);
57                                  ts_A.ts_B_1.ts_C_1.autState = S1;
58
59                          }
60
61                  }
62
63          }
64          if(((std::find(shallow_final_C_2.begin(), shallow_final_C_2.end(),
65          ts_A.ts_B_2.ts_C_2.autState) !=shallow_final_C_2.end() && (S0 == S0 && c <= ts_A.x))
66          || (ts_A.ts_B_2.ts_C_2.autState == S0 && c <= ts_A.x))){
67                  if((std::find(shallow_final_C_2.begin(), shallow_final_C_2.end(),
68                  ts_A.ts_B_2.ts_C_2.autState) !=shallow_final_C_2.end()
69                  && (S0 == S0 && c <= ts_A.x))){
70                          ts_A.ts_B_2.kleeneState = KLEENE_STARTED;
71                          if((S0 == S0 && c <= ts_A.x)){
72                                  Code::alert(c, ts_A.x);
73                                  ts_A.ts_B_2.ts_C_2.autState = S1;
74
75                          }
76
77                  }else if((ts_A.ts_B_2.ts_C_2.autState == S0 && c <= ts_A.x)){
78                          if((ts_A.ts_B_2.ts_C_2.autState == S0 && c <= ts_A.x)){
79                                  Code::alert(c, ts_A.x);
80                                  ts_A.ts_B_2.ts_C_2.autState = S1;
81
82                          }
83
84                  }
85
86          }
87
88  }
```

```
89  int main(int argc, char** argv){
90          ts_A.x = 1;
91          ts_A.ts_B_1.kleeneState = KLEENE_NOTSTARTED;
92          ts_A.ts_B_1.ts_C_1.autState = S0;
93          ts_A.ts_B_2.kleeneState = KLEENE_NOTSTARTED;
94          ts_A.ts_B_2.ts_C_2.autState = S0;
95          while (1){
96                  Event  e = read_event(argc, argv);
97                  if(e.label.compare("e1") == 0){
98                          _safe_(e1(Types::get_int(e.params[0])));
99
100                 }else {
101                         ERROR_1;
102                 }
103
104         }
105         return 1;
106 }
```

B.2 Exemple de génération de code : Cas 2

La figure B.2 montre un exemple de séquence ASTD contenant deux ASTD Kleene. Chaque ASTD Kleene contient un ASTD automate.

C'est le code C ++ généré à partir de la Fig. B.2 par le compilateur ASTD.

```
1  #include "Code.cpp"
2  #include "helper.h"
3  #include "logger.h"
4  enum KleeneState{
5    KLEENE_NOTSTARTED,
6    KLEENE_STARTED
7  };
8  enum AutState{
9    S0,
10   S1
11 };
12 enum SequenceState{
13   FST,
14   SND
15 };
16 struct TState_C_2{
17   AutState  autState;
18
19 };
20 struct TState_B_2{
21   KleeneState  kleeneState;
```

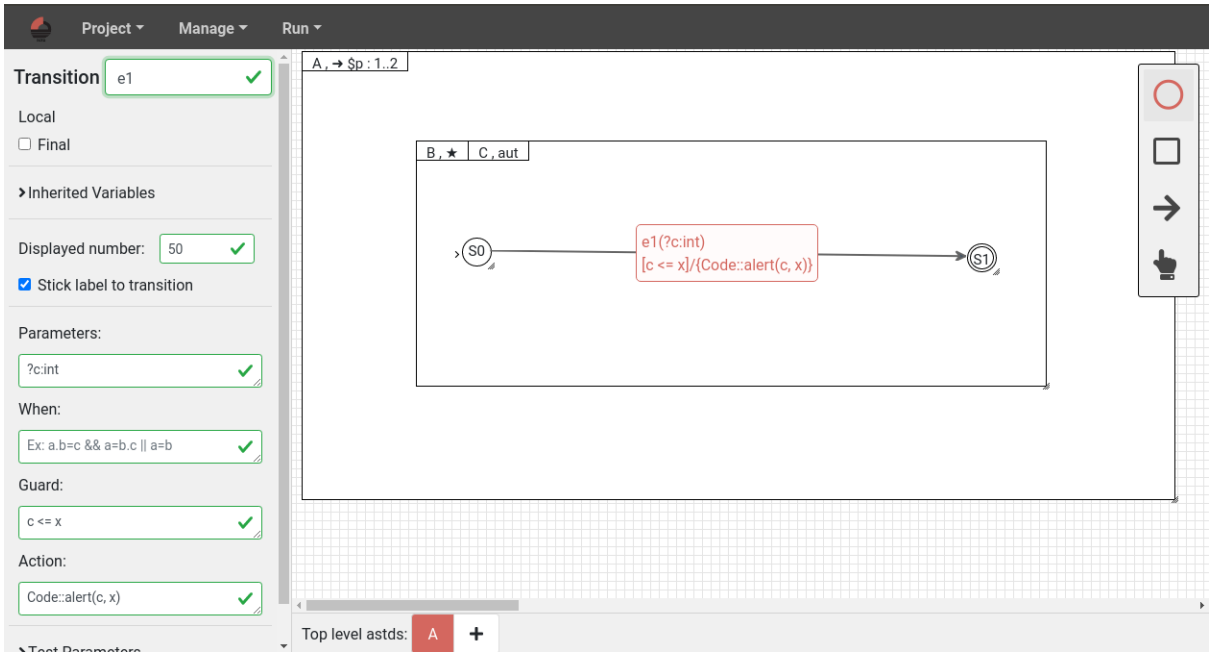


Figure B.2 – Exemple - Séquence, Kleene, Automate

```
22    TState_C_2  ts_C_2;
23
24  };
25  struct TState_C_1{
26    AutState  autState;
27
28  };
29  struct TState_B_1{
30    KleeneState  kleeneState;
31    TState_C_1  ts_C_1;
32
33  };
34  struct TState_A{
35    int  x;
36    SequenceState  sequenceState;
37    TState_B_1  ts_B_1;
38    TState_B_2  ts_B_2;
39
40  };
41  const std::vector<AutState>  shallow_final_C_1 = {S1};
42  const std::vector<AutState>  shallow_final_C_2 = {S1};
43  TState_A  ts_A;
44
45  void e1(int  c){
46      if((ts_A.sequenceState == FST && ((std::find(shallow_final_C_1.begin(),
```

```
47      shallow_final_C_1.end(), ts_A.ts_B_1.ts_C_1.autState) !=shallow_final_C_1.end()
48      && (S0 == S0 && c <= ts_A.x)) || (ts_A.ts_B_1.ts_C_1.autState == S0 && c <= ts_A.x))){
49              if((std::find(shallow_final_C_1.begin(), shallow_final_C_1.end(),
50              ts_A.ts_B_1.ts_C_1.autState) !=shallow_final_C_1.end()
51              && (S0 == S0 && c <= ts_A.x))){
52                      ts_A.ts_B_1.kleeneState = KLEENE_STARTED;
53                      if((S0 == S0 && c <= ts_A.x)){
54                              Code::alert(c, ts_A.x);
55                              ts_A.ts_B_1.ts_C_1.autState = S1;
56
57                      }
58
59              }else if((ts_A.ts_B_1.ts_C_1.autState == S0 && c <= ts_A.x)){
60                      if((ts_A.ts_B_1.ts_C_1.autState == S0 && c <= ts_A.x)){
61                              Code::alert(c, ts_A.x);
62                              ts_A.ts_B_1.ts_C_1.autState = S1;
63
64                      }
65
66              }
67
68      }else if((ts_A.sequenceState == FST && ((ts_A.ts_B_1.kleeneState == KLEENE_NOTSTARTED
69      || std::find(shallow_final_C_1.begin(), shallow_final_C_1.end(),
70      ts_A.ts_B_1.ts_C_1.autState) !=shallow_final_C_1.end()) &&
71      ((std::find(shallow_final_C_2.begin(), shallow_final_C_2.end(), S0)
72      !=shallow_final_C_2.end() && (S0 == S0 && c <= ts_A.x)) || (S0 == S0 && c <= ts_A.x))))){
73              ts_A.sequenceState = SND;
74              if((std::find(shallow_final_C_2.begin(), shallow_final_C_2.end(), S0)
75              !=shallow_final_C_2.end()
76              && (S0 == S0 && c <= ts_A.x))){
77                      ts_A.ts_B_2.kleeneState = KLEENE_STARTED;
78                      if((S0 == S0 && c <= ts_A.x)){
79                              Code::alert(c, ts_A.x);
80                              ts_A.ts_B_2.ts_C_2.autState = S1;
81
82                      }
83
84              }else if((S0 == S0 && c <= ts_A.x)){
85                      if((S0 == S0 && c <= ts_A.x)){
86                              Code::alert(c, ts_A.x);
87                              ts_A.ts_B_2.ts_C_2.autState = S1;
88
89                      }
90
91              }
92
93      }else if((ts_A.sequenceState == SND && ((std::find(shallow_final_C_2.begin(),
94      shallow_final_C_2.end(), ts_A.ts_B_2.ts_C_2.autState) !=shallow_final_C_2.end()
95      && (S0 == S0 && c <= ts_A.x)) || (ts_A.ts_B_2.ts_C_2.autState == S0 && c <= ts_A.x))){
```

```
96              if((std::find(shallow_final_C_2.begin(), shallow_final_C_2.end(),
97              ts_A.ts_B_2.ts_C_2.autState) !=shallow_final_C_2.end()
98              && (S0 == S0 && c <= ts_A.x))){
99                      ts_A.ts_B_2.kleeneState = KLEENE_STARTED;
100                     if((S0 == S0 && c <= ts_A.x)){
101                             Code::alert(c, ts_A.x);
102                             ts_A.ts_B_2.ts_C_2.autState = S1;
103
104                     }
105
106             }else if((ts_A.ts_B_2.ts_C_2.autState == S0 && c <= ts_A.x)){
107                     if((ts_A.ts_B_2.ts_C_2.autState == S0 && c <= ts_A.x)){
108                             Code::alert(c, ts_A.x);
109                             ts_A.ts_B_2.ts_C_2.autState = S1;
110
111                     }
112
113             }
114
115     }
116
117 }
118 int main(int argc, char** argv){
119     ts_A.sequenceState = FST;
120     ts_A.x = 1;
121     ts_A.ts_B_1.kleeneState = KLEENE_NOTSTARTED;
122     ts_A.ts_B_1.ts_C_1.autState = S0;
123     while (1){
124             Event  e = read_event(argc, argv);
125             if(e.label.compare("e1") == 0){
126                     _safe_(e1(Types::get_int(e.params[0])));
127
128             }else {
129                     ERROR_1;
130             }
131
132     }
133     return 1;
134 }
```

B.3 Exemple de génération de code : Cas 3

La figure B.3 montre un exemple de flux ASTD contenant deux ASTD entrelacements quantifiés. Chaque ASTD entrelacement quantifié contient un ASTD Kleene. C'est le code C++ généré à partir de la Fig. B.3 par le compilateur ASTD.

B.3. Exemple de génération de code: Cas 3

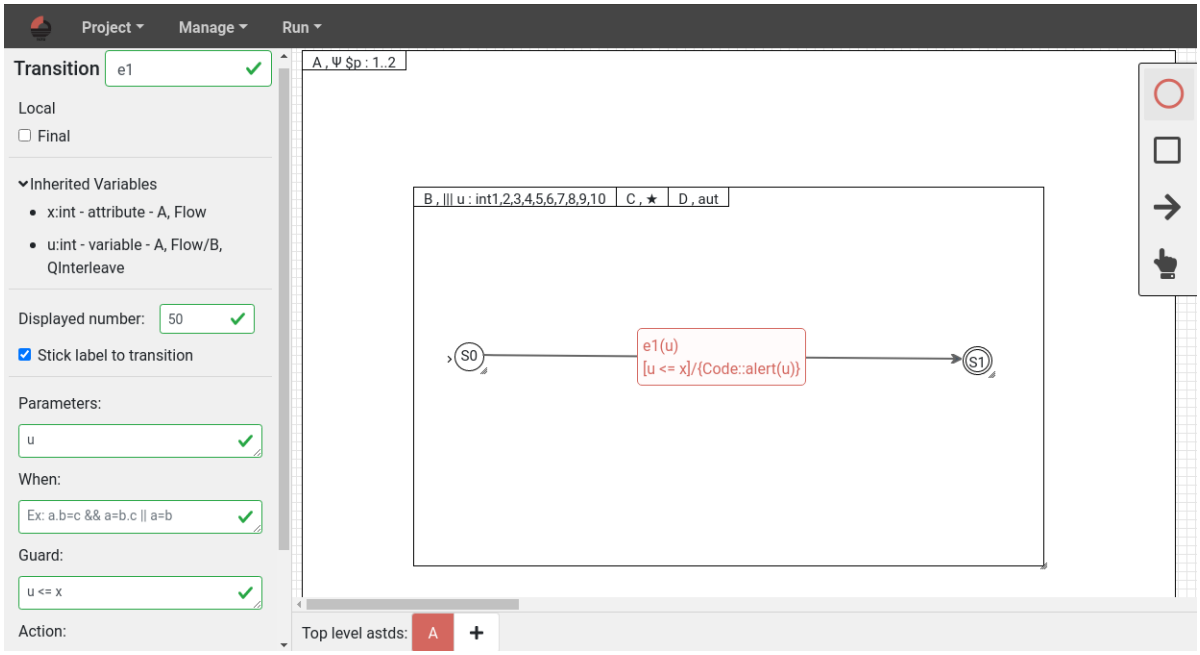


Figure B.3 – Exemple - Flux, QInterleaving, Kleene, Automate

```
#include "Code.cpp"
#include "helper.h"
#include "logger.h"
enum KleeneState{
  KLEENE_NOTSTARTED,
  KLEENE_STARTED
};
enum AutState{
  S0,
  S1
};
struct TState_D_2{
  AutState  autState;

};
struct TState_C_2{
  KleeneState  kleeneState;
  TState_D_2  ts_D_2;

};
struct TState_B_2{
  int  u;
  std::map<int, TState_C_2>  f;

};
```

```
26  struct TState_D_1{
27    AutState  autState;
28
29  };
30  struct TState_C_1{
31    KleeneState  kleeneState;
32    TState_D_1  ts_D_1;
33
34  };
35  struct TState_B_1{
36    int  u;
37    std::map<int, TState_C_1>  f;
38
39  };
40  struct TState_A{
41    int  x;
42    TState_B_1  ts_B_1;
43    TState_B_2  ts_B_2;
44
45  };
46  const std::vector<int>  T_B_1 = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
47  const std::vector<AutState>  shallow_final_D_1 = {S1};
48  const std::vector<int>  T_B_2 = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
49  const std::vector<AutState>  shallow_final_D_2 = {S1};
50  TState_A  ts_A;
51
52  int  exists0(int&  u, int  _u){
53          u = _u;
54          if((std::find(T_B_1.begin(), T_B_1.end(), u) !=T_B_1.end()
55          && ((std::find(shallow_final_D_1.begin(), shallow_final_D_1.end(),
56          ts_A.ts_B_1.f[ts_A.ts_B_2.u].ts_D_1.autState) !=shallow_final_D_1.end() && (S0 == S0
57          && ts_A.ts_B_2.u <= ts_A.x)) || (ts_A.ts_B_1.f[ts_A.ts_B_2.u].ts_D_1.autState == S0
58          && ts_A.ts_B_2.u <= ts_A.x)))){
59                  return 1;
60
61          }
62          return 0;
63
64  }
65
66  void e1(int  _u){
67      if(exists0(ts_A.ts_B_2.u, _u)){
68              if(exists0(ts_A.ts_B_2.u, _u)){
69                      if((std::find(shallow_final_D_1.begin(), shallow_final_D_1.end(),
70                      ts_A.ts_B_1.f[ts_A.ts_B_2.u].ts_D_1.autState) !=shallow_final_D_1.end()
71                      && (S0 == S0 && ts_A.ts_B_2.u <= ts_A.x))){
72                              ts_A.ts_B_1.f[ts_A.ts_B_2.u].kleeneState = KLEENE_STARTED;
73                              if((S0 == S0 && ts_A.ts_B_2.u <= ts_A.x)){
74                                      Code::alert(ts_A.ts_B_2.u);
```

```
75                                      ts_A.ts_B_1.f[ts_A.ts_B_2.u].ts_D_1.autState = S1;
76
77                              }
78
79                      }else if((ts_A.ts_B_1.f[ts_A.ts_B_2.u].ts_D_1.autState == S0
80                      && ts_A.ts_B_2.u <= ts_A.x)){
81                              if((ts_A.ts_B_1.f[ts_A.ts_B_2.u].ts_D_1.autState == S0
82                              && ts_A.ts_B_2.u <= ts_A.x)){
83                                      Code::alert(ts_A.ts_B_2.u);
84                                      ts_A.ts_B_1.f[ts_A.ts_B_2.u].ts_D_1.autState = S1;
85
86                              }
87
88                      }
89
90              }
91
92      }
93      if(exists0(ts_A.ts_B_2.u, _u)){
94              if(exists0(ts_A.ts_B_2.u, _u)){
95                      if((std::find(shallow_final_D_2.begin(), shallow_final_D_2.end(),
96                      ts_A.ts_B_2.f[ts_A.ts_B_2.u].ts_D_2.autState) !=shallow_final_D_2.end()
97                      && (S0 == S0 && ts_A.ts_B_2.u <= ts_A.x))){
98                              ts_A.ts_B_2.f[ts_A.ts_B_2.u].kleeneState = KLEENE_STARTED;
99                              if((S0 == S0 && ts_A.ts_B_2.u <= ts_A.x)){
100                                     Code::alert(ts_A.ts_B_2.u);
101                                     ts_A.ts_B_2.f[ts_A.ts_B_2.u].ts_D_2.autState = S1;
102
103                             }
104
105                     }else if((ts_A.ts_B_2.f[ts_A.ts_B_2.u].ts_D_2.autState == S0
106                     && ts_A.ts_B_2.u <= ts_A.x)){
107                             if((ts_A.ts_B_2.f[ts_A.ts_B_2.u].ts_D_2.autState == S0
108                             && ts_A.ts_B_2.u <= ts_A.x)){
109                                     Code::alert(ts_A.ts_B_2.u);
110                                     ts_A.ts_B_2.f[ts_A.ts_B_2.u].ts_D_2.autState = S1;
111
112                             }
113
114                     }
115
116             }
117
118     }
119
120 }
121 int main(int argc, char** argv){
122         ts_A.x = 1;
123         while (1){
```

```
124                     Event  e = read_event(argc, argv);
125                     if(e.label.compare("e1") == 0){
126                             _safe_(e1(Types::get_int(e.params[0])));
127
128                     }else {
129                             ERROR_1;
130                     }
131             }
132             return 1;
133     }
```

B.4 Exemple d'optimisation de code

Cette section montre l'optimisation du code de l'exemple de la Fig. B.1.

```
1   #include "Code.cpp"
2   #include "helper.h""
3   #include "logger.h"
4   enum KleeneState{
5     KLEENE_NOTSTARTED,
6     KLEENE_STARTED
7   };
8   enum AutState{
9     S0,
10    S1
11  };
12  struct TState_C_2{
13    AutState  autState;
14    int   cond_0;
15
16  };
17  struct TState_B_2{
18    KleeneState  kleeneState;
19    TState_C_2  ts_C_2;
20    int   cond_1;
21    int   cond_2;
22
23  };
24  struct TState_C_1{
25    AutState  autState;
26    int   cond_0;
27
28  };
29  struct TState_B_1{
30    KleeneState  kleeneState;
31    TState_C_1  ts_C_1;
```

```
32    int   cond_1;
33    int   cond_2;
34
35  };
36  struct TState_A{
37    int  x;
38    TState_B_1  ts_B_1;
39    TState_B_2  ts_B_2;
40    int   cond_1;
41    int   cond_2;
42
43  };
44  const std::vector<AutState>  shallow_final_C_1 = {S1};
45  const std::vector<AutState>  shallow_final_C_2 = {S1};
46  TState_A  ts_A;
47
48  void e1(int  c){
49      ts_A.cond_1 = ((ts_A.ts_B_1.cond_1 = ((std::find(shallow_final_C_1.begin(),
50         shallow_final_C_1.end(), ts_A.ts_B_1.ts_C_1.autState) !=shallow_final_C_1.end() &&
51         (ts_A.ts_B_1.ts_C_1.cond_0 = ((ts_A.ts_B_1.ts_C_1.autState == S0
52         && c <= ts_A.x)))))) || (ts_A.ts_B_1.cond_2 = ((ts_A.ts_B_1.ts_C_1.cond_0 =
53         ((ts_A.ts_B_1.ts_C_1.autState == S0 && c <= ts_A.x))))));
54
55      ts_A.cond_2 = ((ts_A.ts_B_2.cond_1 = ((std::find(shallow_final_C_2.begin(),
56         shallow_final_C_2.end(), ts_A.ts_B_2.ts_C_2.autState) !=shallow_final_C_2.end() &&
57         (ts_A.ts_B_2.ts_C_2.cond_0 = ((ts_A.ts_B_2.ts_C_2.autState == S0
58         && c <= ts_A.x)))))) || (ts_A.ts_B_2.cond_2 = ((ts_A.ts_B_2.ts_C_2.cond_0 =
59         ((ts_A.ts_B_2.ts_C_2.autState == S0 && c <= ts_A.x))))));
60
61      if(ts_A.cond_1){
62          if(ts_A.ts_B_1.cond_1){
63                  ts_A.ts_B_1.kleeneState = KLEENE_STARTED;
64                  if((ts_A.ts_B_1.ts_C_1.cond_0 && c <= ts_A.x)){
65                          Code::alert(c, ts_A.x);
66                          ts_A.ts_B_1.ts_C_1.autState = S1;
67
68                  }
69
70          }else if(ts_A.ts_B_1.cond_2){
71                  if((ts_A.ts_B_1.ts_C_1.cond_0 && c <= ts_A.x)){
72                          Code::alert(c, ts_A.x);
73                          ts_A.ts_B_1.ts_C_1.autState = S1;
74
75                  }
76
77          }
78
79      }
80      if(ts_A.cond_2){
```

```
81          if(ts_A.ts_B_2.cond_1){
82                  ts_A.ts_B_2.kleeneState = KLEENE_STARTED;
83                  if((ts_A.ts_B_2.ts_C_2.cond_0 && c <= ts_A.x)){
84                          Code::alert(c, ts_A.x);
85                          ts_A.ts_B_2.ts_C_2.autState = S1;
86
87                  }
88
89          }else if(ts_A.ts_B_2.cond_2){
90                  if((ts_A.ts_B_2.ts_C_2.cond_0 && c <= ts_A.x)){
91                          Code::alert(c, ts_A.x);
92                          ts_A.ts_B_2.ts_C_2.autState = S1;
93
94                  }
95
96          }
97
98      }
99
100 }
101 int main(int argc, char** argv){
102         ts_A.x = 1;
103         ts_A.ts_B_1.kleeneState = KLEENE_NOTSTARTED;
104         ts_A.ts_B_1.ts_C_1.autState = S0;
105         ts_A.ts_B_2.kleeneState = KLEENE_NOTSTARTED;
106         ts_A.ts_B_2.ts_C_2.autState = S0;
107         while (1){
108                 Event  e = read_event(argc, argv);
109                 if(e.label.compare("e1") == 0){
110                         _safe_(e1(Types::get_int(e.params[0])));
111
112                 }else {
113                         ERROR_1;
114                 }
115         }
116         return 1;
117 }
```

Bibliographie

[1] J.-R. Abrial et J.-R. Abrial, *The B-book : assigning programs to meanings*. Cambridge University Press, 2005.

[2] M. H. Ali, B. A. D. Al Mohammed, A. Ismail, et M. F. Zolkipli, « A New Intrusion Detection System Based on Fast Learning Network and Particle Swarm Optimization, » *IEEE Access*, vol. 6, pp. 20 255–20 261, 2018.

[3] J.-R. Abrial, *The B-book : Assigning Programs to Meanings*. New York, NY, USA : Cambridge University Press, 1996.

[4] M. Al-Rubaie et J. M. Chang, « Reconstruction Attacks Against Mobile-Based Continuous Authentication Systems in the Cloud, » *IEEE Transactions on Information Forensics and Security*, vol. 11, no. 12, pp. 2648–2663, Dec 2016.

[5] M. E. Aminanto, R. Choi, H. C. Tanuwidjaja, P. D. Yoo, et K. Kim, « Deep Abstraction and Weighted Feature Selection for Wi-Fi Impersonation Detection, » *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 3, pp. 621–636, March 2018.

[6] U. Aickelin et D. Dasgupta, « Artificial Immune Systems, » *ArXiv e-prints*, octobre 2009.

[7] S. D. Antón, D. Fraunholz, J. Zemitis, F. Pohl, et H. D. Schotten, « Highly Scalable and Flexible Model for Effective Aggregation of Context-based Data in Generic IIoT Scenarios. » dans *ZEUS*, 2017, pp. 51–58.

[8] J. H. Abawajy et M. M. Hassan, « Federated Internet of Things and Cloud Computing Pervasive Patient Health Monitoring System, » *IEEE Communications Magazine*, vol. 55, no. 1, pp. 48–53, January 2017.

[9] D. H. Ackley, G. E. Hinton, et T. J. Sejnowski, « A learning algorithm for Boltzmann machines, » *Cognitive science*, vol. 9, no. 1, pp. 147–169, 1985.

[10] S. Al-Janabi, I. Al-Shourbaji, M. Shojafar, et S. Shamshirband, « Survey of main challenges (security and privacy) in wireless body area networks for healthcare applications, » *Egyptian Informatics Journal*, vol. 18, no. 2, pp. 113 – 122, 2017.

[11] R. Albert, H. Jeong, et A.-L. Barabási, « Error and attack tolerance of complex networks, » *nature*, vol. 406, no. 6794, p. 378, 2000.

[12] J. Abawajy et A. Kelarev, « Iterative Classifier Fusion System for the Detection of Android Malware, » *IEEE Transactions on Big Data*, pp. 1–1, 2018.

[13] M. Al-Qatf, Y. Lasheng, M. Al-Habib, et K. Al-Sabahi, « Deep Learning Approach Combining Sparse Autoencoder With SVM for Network Intrusion Detection, » *IEEE Access*, vol. 6, pp. 52 843–52 856, 2018.

[14] U. Adhikari, T. H. Morris, et S. Pan, « Applying Hoeffding Adaptive Trees for Real-Time Cyber-Power Event and Intrusion Classification, » *IEEE Transactions on Smart Grid*, vol. 9, no. 5, pp. 4049–4060, Sep. 2018.

[15] A. Aamodt et E. Plaza, « Case-based reasoning : Foundational issues, methodological variations, and system approaches, » *AI communications*, vol. 7, no. 1, pp. 39–59, 1994.

[16] R. E. O. Alan Saied et T. Radzik, « Detection of known and unknown DDoS attacks using Artificial Neural Networks, » *Neurocomputing*, vol. 172, no. Supplement C, pp. 385 – 393, 2016.

[17] R. A. R. Ashfaq, X.-Z. Wang, J. Z. Huang, H. Abbas, et Y.-L. He, « Fuzziness based semi-supervised learning approach for intrusion de-

tection system, » *Information Sciences*, vol. 378, no. Supplement C, pp. 484 – 497, 2017.

[18] W. L. Al-Yaseen, Z. A. Othman, et M. Z. A. Nazri, « Multi-level hybrid support vector machine and extreme learning machine based on modified K-means for intrusion detection system, » *Expert Systems with Applications*, vol. 67, pp. 296 – 303, 2017.

[19] I. H. Abdulqadder, D. Zou, I. T. Aziz, B. Yuan, et W. Li, « SecSDN-Cloud : Defeating Vulnerable Attacks Through Secure Software-Defined Networks, » *IEEE Access*, vol. 6, pp. 8292–8301, 2018.

[20] C. M. Bishop *et al.*, *Neural networks for pattern recognition*. Oxford university press, 1995.

[21] Y. Bakhdlaghi, « Snort and SSL/TLS Inspection, » SANS Institute, Rapport technique, 2017, https://www.sans.org/reading-room/whitepapers/detection/snort-ssl-tls-inspection-37735.

[22] S. Baluja, « Population-based incremental learning. a method for integrating genetic search based function optimization and competitive learning, » Carnegie-Mellon Univ Pittsburgh Pa Dept Of Computer Science, Rapport technique, 1994.

[23] H. B. Barlow, « Unsupervised learning, » *Neural computation*, vol. 1, no. 3, pp. 295–311, 1989.

[24] M. H. Bhuyan, D. K. Bhattacharyya, et J. K. Kalita, « Network Anomaly Detection : Methods, Systems and Tools, » *IEEE Communications Surveys Tutorials*, vol. 16, no. 1, pp. 303–336, First 2014.

[25] M. H. Bhuyan, D. K. Bhattacharyya, et J. K. Kalita, « Towards Generating Real-life Datasets for Network Intrusion Detection. » *IJ Network Security*, vol. 17, no. 6, pp. 683–701, 2015.

[26] M. H. Bhuyan, D. Bhattacharyya, et J. K. Kalita, « E-LDAT : a lightweight system for DDoS flooding attack detection and IP traceback using extended entropy metric, » *Security and Communication Networks*, vol. 9, no. 16, pp. 3251–3270, 2016.

[27] J. C. Bezdek, R. Ehrlich, et W. Full, « FCM : The fuzzy c-means clustering algorithm, » *Computers & Geosciences*, vol. 10, no. 2-3, pp. 191–203, 1984.

[28] R. Bellman, « A Markovian Decision Process, » *Indiana Univ. Math. J.*, vol. 6, pp. 679–684, 1957.

[29] H. Barringer, Y. Falcone, K. Havelund, G. Reger, et D. Rydeheard, « Quantified Event Automata : Towards Expressive and Efficient Runtime Monitors, » dans *FM 2012 : Formal Methods*. Springer Berlin Heidelberg, 2012, pp. 68–84.

[30] H. Barringer, Y. Falcone, K. Havelund, G. Reger, et D. Rydeheard, « Quantified Event Automata : Towards Expressive and Efficient Runtime Monitors, » dans *FM 2012 : Formal Methods*, D. Giannakopoulou et D. Méry, éditeurs. Berlin, Heidelberg : Springer Berlin Heidelberg, 2012, pp. 68–84.

[31] M. Barni, P. Failla, R. Lazzeretti, A. Sadeghi, et T. Schneider, « Privacy-Preserving ECG Classification With Branching Programs and Neural Networks, » *IEEE Transactions on Information Forensics and Security*, vol. 6, no. 2, pp. 452–468, June 2011.

[32] A. Bifet et R. Gavalda, « Learning from time-changing data with adaptive windowing, » dans *Proceedings of the 2007 SIAM international conference on data mining*. SIAM, 2007, pp. 443–448.

[33] A. Bifet et R. Gavaldà, « Adaptive learning from evolving data streams, » dans *International Symposium on Intelligent Data Analysis*. Springer, 2009, pp. 249–260.

[34] K. Beyer, J. Goldstein, R. Ramakrishnan, et U. Shaft, « When is nearest neighbor meaningful ? » dans *International conference on database theory*. Springer, 1999, pp. 217–235.

[35] D. Basin, M. Harvan, F. Klaedtke, et E. Zălinescu, « MONPOLY : Monitoring usage-control policies, » dans *International conference on runtime verification*. Springer, 2011, pp. 360–364.

[36] C. M. Bishop, *Pattern recognition and machine learning*. springer, 2006.

[37] S. Behal et K. Kumar, « Detection of DDoS attacks and flash events using novel information theory metrics, » *Computer Networks*, vol. 116, pp. 96–110, 2017.

[38] S. Barakovic, E. Kurtovic, O. Bozanovic, A. Mirojevic, S. Ljevakovic, A. Jokic, M. Peranovic, et J. B. Husic, « Security issues in wireless networks : An overview, » dans *2016 XI International Symposium on Telecommunications (BIHTEL)*, Oct 2016, pp. 1–6.

[39] S. Behal, K. Kumar, et M. Sachdeva, « D-FACE : An anomaly based distributed approach for early detection of DDoS attacks and flash events, » *Journal of Network and Computer Applications*, vol. 111, pp. 49–63, 2018.

[40] T. Bocek, C. Killer, C. Tsiaras, et B. Stiller, *An NFC Relay Attack with Off-the-shelf Hardware and Software*. Cham : Springer International Publishing, 2016, pp. 71–83. Disponible à https://doi.org/10.1007/978-3-319-39814-3_8

[41] M. Butler et M. Leuschel, « Combining CSP and B for Specification and Property Verification, » dans *Proceedings of Formal Methods 2005*, série LNCS 3582. Springer-Verlag, 2005, pp. 221–236.

[42] F. H. Botes, L. Leenen, et R. De La Harpe, « Ant colony induced decision trees for intrusion detection, » dans *ECCWS 2017 16th European Conference on Cyber Warfare and Security*, 2017, p. 53.

[43] D. M. Blei, « Hierarchical clustering, » *Lecture Slides, February*, 2008, http://www.cs.princeton.edu/courses/archive/spr08/cos424/slides/clustering-2.pdf.

[44] F. Bouhafs, M. Mackay, et M. Merabti, « Links to the Future : Communication Requirements and Challenges in the Smart Grid, » *IEEE Power and Energy Magazine*, vol. 10, no. 1, pp. 24–32, Jan 2012.

[45] F. Bonomi, R. Milito, J. Zhu, et S. Addepalli, « Fog computing and its role in the internet of things, » dans *Proceedings of the first edition*

of the MCC workshop on Mobile cloud computing. ACM, 2012, pp. 13–16.

[46] S. L. M. Barrocas et M. Oliveira, « JCircus 2.0 : an Extension of an Automatic Translator from Circus to Java. » dans *CPA*, 2012, pp. 15–36.

[47] B. Brehmer, « The dynamic OODA loop : Amalgamating Boyd's OODA loop and the cybernetic approach to command and control, » dans *Proceedings of the 10th international command and control research technology symposium*, 2005, pp. 365–368.

[48] B. G. Buchanan, E. H. Shortliffe *et al.*, *Rule-based expert systems*. Addison-Wesley Reading, MA, 1984, vol. 3.

[49] E. Börger et R. Stärk, *Abstract state machines : a method for high-level system design and analysis*. Springer Science & Business Media, 2012.

[50] H. Bostani et M. Sheikhan, « Hybrid of binary gravitational search algorithm and mutual information for feature selection in intrusion detection systems, » *Soft Computing*, vol. 21, no. 9, pp. 2307–2324, May 2017. Disponible à https://doi.org/10.1007/s00500-015-1942-8

[51] M. Butler, « csp2B : A Practical Approach to Combining CSP and B, » *Formal Aspects of Computing*, vol. 12, no. 3, pp. 182–198, Nov 2000.

[52] S. B. Baker, W. Xiang, et I. Atkinson, « Internet of Things for Smart Healthcare : Technologies, Challenges, and Opportunities, » *IEEE Access*, vol. 5, pp. 26 521–26 544, 2017.

[53] K. Bu, M. Xu, X. Liu, J. Luo, S. Zhang, et M. Weng, « Deterministic Detection of Cloning Attacks for Anonymous RFID Systems, » *IEEE Transactions on Industrial Informatics*, vol. 11, no. 6, pp. 1255–1266, Dec 2015.

[54] F. Castanedo, « A review of data fusion techniques, » *The Scientific World Journal*, vol. 2013, 2013.

[55] V. Chandola, A. Banerjee, et V. Kumar, « Anomaly Detection : A Survey, » *ACM Comput. Surv.*, vol. 41, no. 3, pp. 15 :1–15 :58, juillet 2009. Disponible à http://doi.acm.org/10.1145/1541880.1541882

[56] K.-C. Chang, C.-Y. Chong, et Y. Bar-Shalom, « Joint probabilistic data association in distributed sensor networks, » *IEEE Transactions on Automatic Control*, vol. 31, no. 10, pp. 889–897, 1986.

[57] P. Cope, J. Campbell, et T. Hayajneh, « An investigation of Bluetooth security vulnerabilities, » dans *2017 IEEE 7th Annual Computing and Communication Workshop and Conference (CCWC)*, Jan 2017, pp. 1–7.

[58] M.-H. Chen, P.-C. Chang, et J.-L. Wu, « A population-based incremental learning approach with artificial immune system for network intrusion detection, » *Engineering Applications of Artificial Intelligence*, vol. 51, pp. 171 – 181, 2016, mining the Humanities : Technologies and Applications.

[59] H. Chen, Y. Fu, et Z. Yan, *Survey on Big Data Analysis Algorithms for Network Security Measurement*. Cham : Springer International Publishing, 2017, pp. 128–142. Disponible à https://doi.org/10.1007/978-3-319-64701-2_10

[60] X. Cao et X. Guo, « Partially Observable Markov Decision Processes With Reward Information : Basic Ideas and Models, » *IEEE Transactions on Automatic Control*, vol. 52, no. 4, pp. 677–681, April 2007.

[61] B. Chandrasekaran, « Generic tasks in knowledge-based reasoning : High-level building blocks for expert system design, » *IEEE expert*, vol. 1, no. 3, pp. 23–30, 1986.

[62] S. Chowdhury, M. Khanzadeh, R. Akula, F. Zhang, S. Zhang, H. Medal, M. Marufuzzaman, et L. Bian, « Botnet detection using graph-based feature clustering, » *Journal of Big Data*, vol. 4, no. 1, p. 14, May 2017. Disponible à https://doi.org/10.1186/s40537-017-0074-7

[63] Q. Chen, R. Luley, Q. Wu, M. Bishop, R. W. Linderman, et Q. Qiu, « AnRAD : A Neuromorphic Anomaly Detection Framework for Massive Concurrent Data Streams, » *IEEE Transactions on Neural Networks and Learning Systems*, vol. 29, no. 5, pp. 1622–1636, May 2018.

[64] Y. Chuan-long, Z. Yue-fei, F. Jin-long, et H. Xin-zheng, « A Deep Learning Approach for Intrusion Detection using Recurrent Neural Networks, » *IEEE Access*, 2017.

[65] C. Chen, S. C. Mukhopadhyay, C. Chuang, T. Lin, M. Liao, Y. Wang, et J. Jiang, « A Hybrid Memetic Framework for Coverage Optimization in Wireless Sensor Networks, » *IEEE Transactions on Cybernetics*, vol. 45, no. 10, pp. 2309–2322, Oct 2015.

[66] F. Cuppens et R. Ortalo, « LAMBDA : A Language to Model a Database for Detection of Attacks, » dans *Recent Advances in Intrusion Detection*. Berlin, Heidelberg : Springer Berlin Heidelberg, 2000, pp. 197–216.

[67] M. Cottrell, M. Olteanu, F. Rossi, et N. Villa-Vialaneix, « Self-OrganizingMaps, theory and applications, » *Revista de Investigacion Operacional*, vol. 39, no. 1, pp. 1–22, 2018.

[68] C. Colombo et G. J. Pace, « Runtime verification using LARVA, » 2017.

[69] O. Chapelle, B. Scholkopf, et A. Zien, « Semi-supervised learning (chapelle, o. et al., eds. ; 2006)[book reviews], » *IEEE Transactions on Neural Networks*, vol. 20, no. 3, pp. 542–542, 2009.

[70] C. Cortes et V. Vapnik, « Support vector machine, » *Machine learning*, vol. 20, no. 3, pp. 273–297, 1995.

[71] P. M. Cao, Y. Wu, S. S. Banerjee, J. Azoff, A. Withers, Z. T. Kalbarczyk, et R. K. Iyer, « {CAUDIT} : Continuous Auditing of {SSH} Servers To Mitigate Brute-Force Attacks, » dans *16th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 19)*, 2019, pp. 667–682.

[72] D. Dasgupta, « Advances in artificial immune systems, » *IEEE Computational Intelligence Magazine*, vol. 1, no. 4, pp. 40–49, Nov 2006.

[73] T. Dash, « A study on intrusion detection using neural networks trained with evolutionary algorithms, » *Soft Computing*, vol. 21, no. 10, pp. 2687–2700, May 2017. Disponible à https://doi.org/10.1007/s00500-015-1967-z

[74] M. Dorigo et C. Blum, « Ant colony optimization theory : A survey, » *Theoretical Computer Science*, vol. 344, no. 2, pp. 243 – 278, 2005.

[75] K. D. et B. B., « On the performance of artificial bee colony (ABC) algorithm, » *Applied soft computing*, vol. 8, no. 1, pp. 687–697, 2008.

[76] A. A. Diro et N. Chilamkurti, « Distributed attack detection scheme using deep learning approach for Internet of Things, » *Future Generation Computer Systems*, 2017.

[77] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, M. Mao, A. Senior, P. Tucker, K. Yang, Q. V. Le *et al.*, « Large scale distributed deep networks, » dans *Advances in neural information processing systems*, 2012, pp. 1223–1231.

[78] D. F. A. L. D. Dean, S. Digrande, « The internet economy in the G20, » Boston Consulting Group, Rapport technique, 2015, www.bcg.com/documents/file100409.pdf.

[79] M. Detyniecki, « Fundamentals on aggregation operators, » *University of California, Berkeley*, 2001.

[80] V. L. Do, L. Fillatre, I. Nikiforov, et P. Willett, « Feature article : security of SCADA systems against cyber-physical attacks, » *IEEE Aerospace and Electronic Systems Magazine*, vol. 32, no. 5, pp. 28–45, May 2017.

[81] J. Dean et S. Ghemawat, « MapReduce : simplified data processing on large clusters, » *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[82] I. Dhyani, N. Goel, G. Sharma, et B. Mallick, *A Reliable Tactic for Detecting Black Hole Attack in Vehicular Ad Hoc Networks*. Singapore : Springer Singapore, 2017, pp. 333–343. Disponible à https://doi.org/10.1007/978-981-10-3770-2_31

[83] N. Decker, J. Harder, T. Scheffel, M. Schmitz, et D. Thoma, « Runtime monitoring with union-find structures, » dans *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2016, pp. 868–884.

[84] R. Devi, R. K. Jha, A. Gupta, S. Jain, et P. Kumar, « Implementation of Intrusion Detection System using Adaptive Neuro-Fuzzy Inference System for 5G wireless communication network, » *AEU - International Journal of Electronics and Communications*, vol. 74, pp. 94 – 106, 2017.

[85] R. B. Diddigi, P. K.J., et S. Bhatnagar, « Novel Sensor Scheduling Scheme for Intruder Tracking in Energy Efficient Sensor Networks, » *IEEE Wireless Communications Letters*, vol. 7, no. 5, pp. 712–715, Oct 2018.

[86] S. Doss, A. Nayyar, G. Suseendran, S. Tanwar, A. Khanna, L. Hoang Son, et P. Huy Thong, « APD-JFAD : Accurate Prevention and Detection of Jelly Fish Attack in MANET, » *IEEE Access*, vol. 6, pp. 56 954–56 965, 2018.

[87] M. D., L. R., S. V., V. V., et A. K. Sangaiah, « Hybrid Reasoning-based Privacy-Aware Disease Prediction Support System, » *Computers Electrical Engineering*, vol. 73, pp. 114 – 127, 2019.

[88] S. Dutta, *Knowledge processing and applied artificial intelligence*. Elsevier, 2014.

[89] S. S. S. R. Depuru, L. Wang, V. Devabhaktuni, et N. Gudi, « Smart meters for power grid - Challenges, issues, advantages and status, » dans *2011 IEEE/PES Power Systems Conference and Exposition*, March 2011, pp. 1–7.

[90] M. Ester, H.-P. Kriegel, J. Sander, X. Xu *et al.*, « A density-based algorithm for discovering clusters in large spatial databases with noise. » dans *Kdd*, vol. 96, no. 34, 1996, pp. 226–231.

[91] S. T. Eckmann, G. Vigna, et R. A. Kemmerer, « STATL : An Attack Language for State-based Intrusion Detection, » *J. Comput. Secur.*, vol. 10, no. 1-2, pp. 71–103, juillet 2002.

[92] T. Fawcett, « An introduction to ROC analysis, » *Pattern Recognition Letters*, vol. 27, no. 8, pp. 861 – 874, 2006, rOC Analysis in Pattern Recognition.

[93] T. Fayolle, « Combinaison des methodes formelles pour la specification de systemes industriels, » Thèse de doctorat, Department of Computer Science, Univ. of Paris Est Creteil, 2005.

[94] B. Fraikin et M. Frappier, « Efficient Symbolic Execution of Large Quantifications in a Process Algebra, » dans *Formal Methods and Software Engineering*. Springer Berlin Heidelberg, 2007, pp. 327–344.

[95] B. Fraikin et M. Frappier, « Efficient symbolic computation of process expressions, » *Science of Computer Programming*, vol. 74, no. 9, pp. 723 – 753, 2009, special Issue on the Fifth International Workshop on Foundations of Coordination Languages and Software Architectures (FOCLASA ?06).

[96] M. Frappier, B. Fraikin, R. Chossart, R. Chane-Yack-Fa, et M. Ouenzar, « Comparison of Model Checking Tools for Information Systems, » dans *Proceedings ICFEM*, 2010, pp. 581–596.

[97] M. Frappier, F. Gervais, R. Laleau, B. Fraikin, et R. St-Denis, « Extending statecharts with process algebra operators, » *Innovations in Systems and Software Engineering*, vol. 4, no. 3, pp. 285–292, Oct 2008.

[98] M. Frappier, F. Gervais, R. Laleau, B. Fraikin, et R. St-Denis, « Extending statecharts with process algebra operators, » *Innovations in Systems and Software Engineering*, vol. 4, no. 3, pp. 285–292, 2008.

[99] M. Frappier, F. Gervais, R. Laleau, et J. Milhau, « Refinement patterns for ASTDs, » *Formal Aspects of Computing*, vol. 26, no. 5, pp. 919–941, Sep 2014.

[100] M. Frappier, F. Gervais, R. Laleau, et J. Milhau, « Refinement patterns for ASTDs, » *Formal Aspects of Computing*, vol. 26, no. 5, pp. 919–941, 2014.

[101] C. Feng, T. Li, Z. Zhu, et D. Chana, « A Deep Learning-based Framework for Conducting Stealthy Attacks in Industrial Control Systems, » *CoRR*, vol. abs/1709.06397, 2017. Disponible à http://arxiv.org/abs/1709.06397

[102] M. Faezipour, M. Nourani, A. Saeed, et S. Addepalli, « Progress and Challenges in Intelligent Vehicle Area Networks, » *Commun. ACM*, vol. 55, no. 2, pp. 90–100, février 2012. Disponible à http://doi.acm.org/10.1145/2076450.2076470

[103] Formal Systems (Europe) Ltd, *Failures-Divergence Refinement — FDR User Manual (version 4.2)*. Disponible à https://www.cs.ox.ac.uk/projects/fdr/manual/

[104] L. Fernandez Maima, A. L. Perales Gomez, F. J. Garcia Clemente, M. Gil Perez, et G. Martinez Perez, « A Self-Adaptive Deep Learning-Based System for Anomaly Detection in 5G Networks, » *IEEE Access*, vol. 6, pp. 7700–7712, 2018.

[105] K. Fukushima, « Neocognitron : A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position, » *Biological cybernetics*, vol. 36, no. 4, pp. 193–202, 1980.

[106] C. J. Fung et Q. Zhu, « FACID : A trust-based collaborative decision framework for intrusion detection networks, » *Ad Hoc Networks*, vol. 53, no. Supplement C, pp. 17 – 31, 2016.

[107] S. R. Garner *et al.*, « Weka : The waikato environment for knowledge analysis, » dans *Proceedings of the New Zealand computer science research students conference*, 1995, pp. 57–64.

[108] M. Gągolewski, *Data fusion : theory, methods, and applications*. Institute of Computer Science Polish Academy of Sciences, 2015.

[109] B. Gaines, « Foundations of fuzzy reasoning, » *International Journal of Man-Machine Studies*, vol. 8, no. 6, pp. 623 – 668, 1976.

[110] J. Goh, S. Adepu, K. N. Junejo, et A. Mathur, *A Dataset to Support Research in the Design of Secure Water Treatment Systems*. Cham : Springer International Publishing, 2017, pp. 88–99. Disponible à https://doi.org/10.1007/978-3-319-71368-7_8

[111] F. Galton, « Regression towards mediocrity in hereditary stature. » *The Journal of the Anthropological Institute of Great Britain and Ireland*, vol. 15, pp. 246–263, 1886.

[112] J. Gama, *Knowledge discovery from data streams*. Chapman and Hall/-CRC, 2010.

[113] S. M. Garrett, « How do we evaluate artificial immune systems ? » *Evolutionary computation*, vol. 13, no. 2, pp. 145–177, 2005.

[114] X. Glorot et Y. Bengio, « Understanding the difficulty of training deep feedforward neural networks, » dans *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, 2010, pp. 249–256.

[115] S. García, M. Grill, J. Stiborek, et A. Zunino, « An empirical comparison of botnet detection methods, » *Computers & Security*, vol. 45, pp. 100 – 123, 2014.

[116] L. Gupta, R. Jain, et G. Vaszkun, « Survey of important issues in UAV communication networks, » *IEEE Communications Surveys & Tutorials*, vol. 18, no. 2, pp. 1123–1152, 2016.

[117] S. Garg, K. Kaur, N. Kumar, et J. J. Rodrigues, « Hybrid Deep Learning-based Anomaly Detection Scheme for Suspicious Flow Detection in SDN : A Social Multimedia Perspective, » *IEEE Transactions on Multimedia*, pp. 1–1, 2019.

[118] J. Gama, P. Medas, G. Castillo, et P. Rodrigues, « Learning with drift detection, » dans *Brazilian symposium on artificial intelligence*. Springer, 2004, pp. 286–295.

[119] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, et Y. Bengio, « Generative adversarial nets, » dans *Advances in neural information processing systems*, 2014, pp. 2672–2680.

[120] K. Grosse, N. Papernot, P. Manoharan, M. Backes, et P. McDaniel, « Adversarial examples for malware detection, » dans *European Symposium on Research in Computer Security*. Springer, 2017, pp. 62–79.

[121] L. Graham, « Cybercrime costs the global economy $450 billion : CEO, » https ://www.cnbc.com/2017/02/07/cybercrime-costs-

the-global-economy-450-billion-ceo.html, Feb. 2017, onlin, accessed 08/11/17.

[122] R. Graham, « Zero Overhead Networking, » *The International Journal of PoC||GTFO*, pp. 66–75, 2017, https://www.alchemistowl.org/pocorgtfo/pocorgtfo15.pdf.

[123] R. A. Grimes, « Danger : Remote Access Trojans, » https ://technet.microsoft.com/en-us/library/dd632947.aspx, Apr. 2018, online, accessed 09/04/18.

[124] K. R. Gariga, A. R. M. Reddy, et N. S. Rao, *PDA-CS : Profile Distance Assessment-Centric Cuckoo Search for Anomaly-Based Intrusion Detection in High-Speed Networks*. Singapore : Springer Singapore, 2017, pp. 169–179. Disponible à https://doi.org/10.1007/978-981-10-3153-3_17

[125] A. J. Galloway et W. J. Stoddart, « An operational semantics for ZCCS, » dans *First IEEE International Conference on Formal Engineering Methods*, Nov 1997, pp. 272–282.

[126] C. V. Goldman et S. Zilberstein, « Optimizing information exchange in cooperative multi-agent systems, » dans *Proceedings of the second international joint conference on Autonomous agents and multiagent systems*. ACM, 2003, pp. 137–144.

[127] G. E. Hinton *et al.*, « Learning distributed representations of concepts, » dans *Proceedings of the eighth annual conference of the cognitive science society*, vol. 1. Amherst, MA, 1986, p. 12.

[128] S. Hansche, *Official (ISC)2 Guide to the ISSEP CBK*, 2nd édition. Boston, MA, USA : Auerbach Publications, 2013.

[129] J. A. Hartigan, « Statistical theory in clustering, » *Journal of classification*, vol. 2, no. 1, pp. 63–76, 1985.

[130] D. Harel, « Statecharts : A visual formalism for complex systems, » *Science of computer programming*, vol. 8, no. 3, pp. 231–274, 1987.

[131] D. He, S. Chan, et M. Guizani, « Drone-Assisted Public Safety Networks : The Security Aspect, » *IEEE Communications Magazine*, vol. 55, no. 8, pp. 218–223, 2017.

[132] J.-P. Hubaux, S. Capkun, et J. Luo, « The security and privacy of smart vehicles, » *IEEE Security & Privacy*, no. 3, pp. 49–55, 2004.

[133] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, et I. H. Witten, « The WEKA data mining software : an update, » *ACM SIGKDD explorations newsletter*, vol. 11, no. 1, pp. 10–18, 2009.

[134] W. Haider, J. Hu, J. Slay, B. Turnbull, et Y. Xie, « Generating realistic intrusion detection system dataset based on fuzzy qualitative modeling, » *Journal of Network and Computer Applications*, vol. 87, no. Supplement C, pp. 185 – 192, 2017.

[135] G. E. Hinton, « Distributed representations, » 1984.

[136] G. E. Hinton, « How neural networks learn from experience, » *Scientific American*, vol. 267, no. 3, pp. 144–151, 1992.

[137] H. Haddad Pajouh, R. Javidan, R. Khayami, D. Ali, et K. R. Choo, « A Two-layer Dimension Reduction and Two-tier Classification Model for Anomaly-Based Intrusion Detection in IoT Backbone Networks, » *IEEE Transactions on Emerging Topics in Computing*, pp. 1–1, 2018.

[138] S. Halle, R. Khoury, et M. Awesso, « Streamlining the Inclusion of Computer Experiments In a Research Paper, » *Computer*, vol. 51, no. 11, pp. 78–89, 2018.

[139] D. L. Hall et J. Llinas, « An introduction to multisensor data fusion, » *Proceedings of the IEEE*, vol. 85, no. 1, pp. 6–23, 1997.

[140] D. L. Hall et S. A. McMullen, *Mathematical techniques in multisensor data fusion*. Artech House, 2004.

[141] D. Harel et A. Naamad, « The STATEMATE Semantics of Statecharts, » *ACM Trans. Softw. Eng. Methodol.*, vol. 5, no. 4, pp. 293–333, octobre 1996.

[142] C. A. R. Hoare, « Communicating Sequential Processes, » *Commun. ACM*, vol. 21, no. 8, pp. 666–677, août 1978.

[143] M. Hashem Eiza, T. Owens, et Q. Ni, « Secure and Robust Multi-Constrained QoS Aware Routing Algorithm for VANETs, » *IEEE Transactions on Dependable and Secure Computing*, vol. 13, no. 1, pp. 32–45, Jan 2016.

[144] I. Horrocks, P. F. Patel-Schneider, H. Boley, S. Tabet, B. Grosof, M. Dean *et al.*, « SWRL : A semantic web rule language combining OWL and RuleML, » *W3C Member submission*, vol. 21, p. 79, 2004.

[145] G. Hamon et J. Rushby, « An operational semantics for Stateflow, » dans *International Conference on Fundamental Approaches to Software Engineering*. Springer, 2004, pp. 229–243.

[146] S. Hochreiter et J. Schmidhuber, « Long short-term memory, » *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[147] G. E. Hinton et R. R. Salakhutdinov, « Reducing the dimensionality of data with neural networks, » *science*, vol. 313, no. 5786, pp. 504–507, 2006.

[148] H. Hasrouny, A. E. Samhat, C. Bassil, et A. Laouiti, « VANet security challenges and solutions : A survey, » *Vehicular Communications*, vol. 7, no. Supplement C, pp. 7 – 20, 2017.

[149] B. Hammer et T. Villmann, « Generalized relevance learning vector quantization, » *Neural Networks*, vol. 15, no. 8-9, pp. 1059–1068, 2002.

[150] G. E. Hinton et R. S. Zemel, « Autoencoders, minimum description length and Helmholtz free energy, » dans *Advances in neural information processing systems*, 1994, pp. 3–10.

[151] G.-B. Huang, Q.-Y. Zhu, et C.-K. Siew, « Extreme learning machine : theory and applications, » *Neurocomputing*, vol. 70, no. 1, pp. 489–501, 2006.

[152] K. Huang, Q. Zhang, C. Zhou, N. Xiong, et Y. Qin, « An Efficient Intrusion Detection Approach for Visual Sensor Networks Based on Traffic Pattern Learning, » *IEEE Transactions on Systems, Man, and Cybernetics : Systems*, vol. 47, no. 10, pp. 2704–2713, Oct 2017.

[153] S. Iqbal, M. L. M. Kiah, B. Dhaghighi, M. Hussain, S. Khan, M. K. Khan, et K.-K. R. Choo, « On cloud security attacks : A taxonomy and intrusion detection and prevention as a service, » *Journal of Network and Computer Applications*, vol. 74, no. Supplement C, pp. 98 – 120, 2016.

[154] H. Iba et N. Noman, *Evolutionary Computation in Gene Regulatory Network Research*. John Wiley & Sons, 2016.

[155] iASTD repository, « Université de Sherbrooke, » https://depot.gril.usherbrooke.ca/fram1801/iASTD-public, 2019.

[156] P. Jackson, « Introduction to expert systems, » 1986.

[157] A. K. Jain, R. C. Dubes *et al.*, *Algorithms for clustering data*. Prentice hall Englewood Cliffs, 1988, vol. 6.

[158] P. Jokar et V. C. M. Leung, « Intrusion Detection and Prevention for ZigBee-Based Home Area Networks in Smart Grids, » *IEEE Transactions on Smart Grid*, vol. 9, no. 3, pp. 1800–1811, May 2018.

[159] X. Jin, J. Liang, W. Tong, L. Lu, et Z. Li, « Multi-agent trust-based intrusion detection scheme for wireless sensor networks, » *Computers & Electrical Engineering*, vol. 59, pp. 262 – 273, 2017.

[160] S. C. Johnson, « Hierarchical clustering schemes, » *Psychometrika*, vol. 32, no. 3, pp. 241–254, 1967.

[161] C. B. Jones, *Systematic software development using VDM*. Citeseer, 1990, vol. 2.

[162] W. Jiang, H. Song, et Y. Dai, « Real-time intrusion detection for high-speed networks, » *Computers & security*, vol. 24, no. 4, pp. 287–294, 2005.

[163] O. Kaiwartya, A. H. Abdullah, Y. Cao, A. Altameem, M. Prasad, C. Lin, et X. Liu, « Internet of Vehicles : Motivation, Layered Architecture, Network Model, Challenges, and Future Aspects, » *IEEE Access*, vol. 4, pp. 5356–5373, 2016.

[164] G. Karagiannis, O. Altintas, E. Ekici, G. Heijenk, B. Jarupan, K. Lin, et T. Weil, « Vehicular networking : A survey and tutorial on requirements, architectures, challenges, standards and solutions, » *IEEE communications surveys & tutorials*, vol. 13, no. 4, pp. 584–616, 2011.

[165] M. Karwaski, « Efficiently Deducing IDS False Positives Using System Profiling, » SANS Institute, Rapport technique, 2009, https://www.sans.org/reading-room/whitepapers/intrusion/efficiently-deducing-ids-false-positives-system-profiling-33223.

[166] D. Karaboga et B. Basturk, « A powerful and efficient algorithm for numerical function optimization : artificial bee colony (ABC) algorithm, » *Journal of Global Optimization*, vol. 39, no. 3, pp. 459–471, Nov 2007. Disponible à https://doi.org/10.1007/s10898-007-9149-x

[167] J. Kennedy, « Particle swarm optimization, » *Encyclopedia of machine learning*, pp. 760–766, 2010.

[168] D. Kotz, C. A. Gunter, S. Kumar, et J. P. Weiner, « Privacy and Security in Mobile Health : A Research Agenda, » *Computer*, vol. 49, no. 6, pp. 22–30, June 2016.

[169] F. A. Khan, N. A. H. Haldar, A. Ali, M. Iftikhar, T. A. Zia, et A. Y. Zomaya, « A Continuous Change Detection Mechanism to Identify Anomalies in ECG Signals for WBAN-Based Healthcare Environments, » *IEEE Access*, vol. 5, pp. 13 531–13 544, 2017.

[170] M. Korczynski, A. Hamieh, J. H. Huh, H. Holm, S. R. Rajagopalan, et N. H. Fefferman, « Hive oversight for network intrusion early warning using DIAMoND : a bee-inspired method for fully distributed cyber defense, » *IEEE Communications Magazine*, vol. 54, no. 6, pp. 60–67, June 2016.

[171] H. Khurana, M. Hadley, N. Lu, et D. A. Frincke, « Smart-grid security issues, » *IEEE Security & Privacy*, vol. 8, no. 1, pp. 81–85, 2010.

[172] J. Kevric, S. Jukic, et A. Subasi, « An effective combining classifier approach using tree algorithms for network intrusion

detection, » *Neural Computing and Applications*, Jun 2016. Disponible à https://doi.org/10.1007/s00521-016-2418-1

[173] B. Karp et H.-T. Kung, « GPSR : Greedy perimeter stateless routing for wireless networks, » dans *Proceedings of the 6th annual international conference on Mobile computing and networking*. ACM, 2000, pp. 243–254.

[174] D. Kwon, H. Kim, D. An, et H. Ju, « DDoS attack volume forecasting using a statistical approach, » dans *2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, May 2017, pp. 1083–1086.

[175] T. Kim, B. Kang, M. Rho, S. Sezer, et E. G. Im, « A Multimodal Deep Learning Method for Android Malware Detection Using Various Features, » *IEEE Transactions on Information Forensics and Security*, vol. 14, no. 3, pp. 773–788, March 2019.

[176] C. Kolias, G. Kambourakis, A. Stavrou, et S. Gritzalis, « Intrusion Detection in 802.11 Networks : Empirical Evaluation of Threats and a Public Dataset, » *IEEE Communications Surveys Tutorials*, vol. 18, no. 1, pp. 184–208, Firstquarter 2016.

[177] T. R. B. Kushal, K. Lai, et M. S. Illindala, « Risk-based Mitigation of Load Curtailment Cyber Attack Using Intelligent Agents in a Shipboard Power System, » *IEEE Transactions on Smart Grid*, pp. 1–1, 2018.

[178] T. Kohonen, « The self-organizing map, » *Proceedings of the IEEE*, vol. 78, no. 9, pp. 1464–1480, Sep 1990.

[179] J. Kolodner, *Case-based reasoning*. Morgan Kaufmann, 2014.

[180] H.-P. Kriegel et M. Pfeifle, « Density-based clustering of uncertain data, » dans *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*. ACM, 2005, pp. 672–677.

[181] J. D. Kalbfleisch et R. L. Prentice, *The statistical analysis of failure time data*. John Wiley & Sons, 2011, vol. 360.

[182] N. Komninos, E. Philippou, et A. Pitsillides, « Survey in Smart Grid and Smart Home Security : Issues, Challenges and Countermeasures, » *IEEE Communications Surveys Tutorials*, vol. 16, no. 4, pp. 1933–1954, Fourthquarter 2014.

[183] M. Kranz, « Fog Analytics : Turning Data into Real-Time Insight and Action, » https ://blogs.cisco.com/digital/fog-analytics-turning-data-into-real-time-insight-and-action, Aug. 2015, online, accessed 12/19/17.

[184] S. Kurt, H. U. Yildiz, M. Yigit, B. Tavli, et V. C. Gungor, « Packet Size Optimization in Wireless Sensor Networks for Smart Grid Applications, » *IEEE Transactions on Industrial Electronics*, vol. 64, no. 3, pp. 2392–2401, March 2017.

[185] S. B. Kotsiantis, I. Zaharakis, et P. Pintelas, « Supervised machine learning : A review of classification techniques, » *Emerging artificial intelligence applications in computer engineering*, vol. 160, pp. 3–24, 2007.

[186] L. Lamport, « Time, Clocks, and the Ordering of Events in a Distributed System, » *Commun. ACM*, vol. 21, no. 7, pp. 558–565, juillet 1978. Disponible à https://doi.org/10.1145/359545.359563

[187] M. Leuschel et M. J. Butler, « ProB : an automated analysis toolset for the B method, » *STTT*, vol. 10, no. 2, pp. 185–203, 2008.

[188] M. Leuschel et E. Börger, « A Compact Encoding of Sequential ASMs in Event-B, » dans *Proceedings ABZ 2016*, 2016, pp. 119–134.

[189] Y. LeCun, Y. Bengio, et G. Hinton, « Deep learning, » *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[190] D. D. Lewis, « Naive (Bayes) at forty : The independence assumption in information retrieval, » dans *European conference on machine learning*. Springer, 1998, pp. 4–15.

[191] Y. LeCun, P. Haffner, L. Bottou, et Y. Bengio, « Object recognition with gradient-based learning, » dans *Shape, contour and grouping in computer vision*. Springer, 1999, pp. 319–345.

[192] C. Lochert, H. Hartenstein, J. Tian, H. Fussler, D. Hermann, et M. Mauve, « A routing strategy for vehicular ad hoc networks in city environments, » dans *IEEE IV2003 Intelligent Vehicles Symposium. Proceedings (Cat. No. 03TH8683)*. IEEE, 2003, pp. 156–161.

[193] P. Likarish et E. Jung, « A targeted web crawling for building malicious javascript collection, » dans *Proceedings of the ACM first international workshop on Data-intensive software management and mining*. ACM, 2009, pp. 23–26.

[194] H. Larochelle et I. Murray, « The neural autoregressive distribution estimator, » dans *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, 2011, pp. 29–37.

[195] W. Li, W. Meng, L.-F. Kwok, et H. H. IP, « Enhancing collaborative intrusion detection networks against insider attacks using supervised intrusion sensitivity-based trust management model, » *Journal of Network and Computer Applications*, vol. 77, no. Supplement C, pp. 135 – 145, 2017.

[196] W. Li, W. Meng, C. Su, et L. F. Kwok, « Towards False Alarm Reduction Using Fuzzy If-Then Rules for Medical Cyber Physical Systems, » *IEEE Access*, vol. 6, pp. 6530–6539, 2018.

[197] A. M. L. Nganyewou Tidjon, M. Frappier, « Technical Report 25, » University of Sherbrooke, Rapport technique, 2018.

[198] Z. Li, Z. Qin, K. Huang, X. Yang, et S. Ye, « Intrusion Detection Using Convolutional Neural Networks for Representation Learning, » dans *International Conference on Neural Information Processing*. Springer, 2017, pp. 858–866.

[199] M. Lu et J. Reeves, « Types of Cyber attacks, » Trustworthy Cyber Infrastructure for the Power Grid (TCIPG), Rapport technique, 2018, https://tcipg.org/sites/default/files/rgroup/tcipg-reading-group-fall_2014_09-12.pdf.

[200] T. Li, J. Ren, et X. Tang, « Secure wireless monitoring and control systems for smart grid and smart home, » *IEEE Wireless Communications*, vol. 19, no. 3, pp. 66–73, June 2012.

[201] M. Leucker et C. Schallhart, « A brief account of runtime verification, » *The Journal of Logic and Algebraic Programming*, vol. 78, no. 5, pp. 293 – 303, 2009, the 1st Workshop on Formal Languages and Analysis of Contract-Oriented Software (FLACOS'07).

[202] F. Y. Leu, K. L. Tsai, Y. T. Hsiao, et C. T. Yang, « An Internal Intrusion Detection and Protection System by Using Data Mining and Forensic Techniques, » *IEEE Systems Journal*, vol. 11, no. 2, pp. 427–438, June 2017.

[203] D. C. Luckham, *Event processing for business : organizing the real-time enterprise*. John Wiley & Sons, 2011.

[204] G. Liang, S. R. Weller, F. Luo, J. Zhao, et Z. Y. Dong, « Generalized FDIA-Based Cyber Topology Attack With Application to the Australian Electricity Market Trading Mechanism, » *IEEE Transactions on Smart Grid*, vol. 9, no. 4, pp. 3820–3829, July 2018.

[205] W. Lin, L. Xiang, D. Pao, et B. Liu, « Collaborative Distributed Intrusion Detection System, » dans *2008 Second International Conference on Future Generation Communication and Networking*, vol. 1, Dec 2008, pp. 172–177.

[206] J. Lin, W. Yu, N. Zhang, X. Yang, et L. Ge, « On data integrity attacks against route guidance in transportation-based cyber-physical systems, » dans *2017 14th IEEE Annual Consumer Communications Networking Conference (CCNC)*, Jan 2017, pp. 313–318.

[207] J. Lin, W. Yu, N. Zhang, X. Yang, H. Zhang, et W. Zhao, « A Survey on Internet of Things : Architecture, Enabling Technologies, Security and Privacy, and Applications, » *IEEE Internet of Things Journal*, vol. 4, no. 5, pp. 1125–1142, Oct 2017.

[208] G. Liang, J. Zhao, F. Luo, S. R. Weller, et Z. Y. Dong, « A Review of False Data Injection Attacks Against Modern Power Systems, » *IEEE Transactions on Smart Grid*, vol. 8, no. 4, pp. 1630–1638, July 2017.

[209] M. N. Mejri, J. Ben-Othman, et M. Hamdi, « Survey on VANET security challenges and possible cryptographic solutions, » *Vehicular Communications*, vol. 1, no. 2, pp. 53–66, 2014.

[210] P. Moscato et C. Cotta, « A gentle introduction to memetic algorithms, » dans *Handbook of metaheuristics*. Springer, 2003, pp. 105–144.

[211] R. Mitchell et I. Chen, « Behavior Rule Specification-Based Intrusion Detection for Safety Critical Medical Cyber Physical Systems, » *IEEE Transactions on Dependable and Secure Computing*, vol. 12, no. 1, pp. 16–30, Jan 2015.

[212] W. S. McCulloch, *Embodiments of mind*. mit Press, 1965.

[213] N. Moustafa, K. R. Choo, I. Radwan, et S. Camtepe, « Outlier Dirichlet Mixture Mechanism : Adversarial Statistical Learning for Anomaly Detection in the Fog, » *IEEE Transactions on Information Forensics and Security*, pp. 1–1, 2019.

[214] B. Morin et H. Debar, « Correlation of Intrusion Symptoms : An Application of Chronicles, » dans *Recent Advances in Intrusion Detection*. Springer Berlin Heidelberg, 2003, pp. 94–112.

[215] N. Marchang, R. Datta, et S. K. Das, « A Novel Approach for Efficient Usage of Intrusion Detection System in Mobile Ad Hoc Networks, » *IEEE Transactions on Vehicular Technology*, vol. 66, no. 2, pp. 1684–1695, Feb 2017.

[216] J. Milhau, M. Frappier, F. Gervais, et R. Laleau, « Systematic Translation Rules from ASTD to Event-B, » dans *Integrated Formal Methods*. Springer Berlin Heidelberg, 2010, pp. 245–259.

[217] J. Milhau, M. Frappier, F. Gervais, et R. Laleau, « Systematic Translation Rules from astd to Event-B, » dans *Integrated Formal Methods*,

D. Méry et S. Merz, éditeurs. Berlin, Heidelberg : Springer Berlin Heidelberg, 2010, pp. 245–259.

[218] M. Mylrea et S. N. G. Gourisetti, *Cybersecurity and Optimization in Smart "Autonomous" Buildings*. Cham : Springer International Publishing, 2017, pp. 263–294. Disponible à https://doi.org/10.1007/978-3-319-59719-5_12

[219] T. M. Mitchell, « Generative and Discriminative classifiers : Naive Bayes and Logistic Regression, » Carnegie Mellon University, Rapport technique, Feb. 2016, https://www.cs.cmu.edu/~tom/mlbook/NBayesLogReg.pdf.

[220] Z. MacHardy, A. Khan, K. Obana, et S. Iwashina, « V2X Access Technologies : Regulation, Research, and Remaining Challenges, » *IEEE Communications Surveys Tutorials*, vol. 20, no. 3, pp. 1858–1877, third-quarter 2018.

[221] P. McDaniel et S. McLaughlin, « Security and Privacy Challenges in the Smart Grid, » *IEEE Security Privacy*, vol. 7, no. 3, pp. 75–77, May 2009.

[222] R. Möller et B. Neumann, « Ontology-based reasoning techniques for multimedia interpretation and retrieval, » dans *Semantic multimedia and ontologies*. Springer, 2008, pp. 55–98.

[223] P. Mishra, E. S. Pilli, V. Varadharajan, et U. Tupakula, « Intrusion detection techniques in cloud environment : A survey, » *Journal of Network and Computer Applications*, vol. 77, no. Supplement C, pp. 18 – 47, 2017.

[224] W. S. Mark et R. L. Simpson, « Knowledge-based systems : an overview, » *IEEE Expert*, vol. 6, no. 3, pp. 12–17, June 1991.

[225] F. Moller et C. Tofts, « A temporal calculus of communicating systems, » dans *International Conference on Concurrency Theory*. Springer, 1990, pp. 401–415.

[226] L. Mechtri, F. D. Tolba, et S. Ghanemi, « An optimized intrusion response system for MANET :, » *Peer-to-*

Peer Networking and Applications, Jun 2017. Disponible à https://doi.org/10.1007/s12083-017-0573-5

[227] R. B. Myerson, *Game theory*. Harvard university press, 2013.

[228] R. Martin, J. Zhang, C. Liu *et al.*, « Dempster–Shafer theory and statistical inference with weak beliefs, » *Statistical Science*, vol. 25, no. 1, pp. 72–87, 2010.

[229] Y. Meng, W. Zhang, H. Zhu, et X. S. Shen, « Securing Consumer IoT in the Smart Home : Architecture, Challenges, and Countermeasures, » *IEEE Wireless Communications*, vol. 25, no. 6, pp. 53–59, December 2018.

[230] L. Nishani et M. Biba, « Machine learning for intrusion detection in MANET : a state-of-the-art survey, » *Journal of Intelligent Information Systems*, vol. 46, no. 2, pp. 391–407, Apr 2016. Disponible à https://doi.org/10.1007/s10844-015-0387-y

[231] N. Nissim, A. Cohen, et Y. Elovici, « ALDOCX : Detection of Unknown Malicious Microsoft Office Documents Using Designated Active Learning Methods Based on New Structural Feature Extraction Methodology, » *IEEE Transactions on Information Forensics and Security*, vol. 12, no. 3, pp. 631–646, March 2017.

[232] N. Naik, R. Diao, et Q. Shen, « Dynamic Fuzzy Rule Interpolation and Its Application to Intrusion Detection, » *IEEE Transactions on Fuzzy Systems*, vol. 26, no. 4, pp. 1878–1892, Aug 2018.

[233] L. H. Newman, « Medical Devices Are the Next Security Nightmare, » https ://www.wired.com/2017/03/medical-devices-next-security-nightmare/, March 2017, online, accessed 09/23/17.

[234] L. Nganyewou Tidjon, M. Frappier, M. Leuschel, et A. Mammar, « Extended Algebraic State-Transition Diagrams, » dans *2018 23rd International Conference on Engineering of Complex Computer Systems (ICECCS)*, Dec 2018, pp. 146–155.

[235] L. Nganyewou Tidjon, M. Frappier, M. Leuschel, et A. Mammar, « Extended Algebraic State-Transition Diagrams, » dans *2018 23rd Inter-*

national Conference on Engineering of Complex Computer Systems (ICECCS), Melbourne, VIC, 2018, pp. 146–155.

[236] S. Noel, E. Harley, K. Tam, M. Limiero, et M. Share, « Chapter 4 - CyGraph : Graph-Based Analytics and Visualization for Cybersecurity, » dans *Cognitive Computing : Theory and Applications*, série Handbook of Statistics, V. N. Gudivada, V. V. Raghavan, V. Govindaraju, et C. Rao, éditeurs. Elsevier, 2016, vol. 35, no. Supplement C, pp. 117 – 167.

[237] N. J. Nilsson, « Learning machines. » Mcgraw-Hill, New York, 1965.

[238] N. Nissim, R. Moskovitch, O. BarAd, L. Rokach, et Y. Elovici, « ALDROID : efficient update of Android anti-virus software using designated active learning methods, » *Knowledge and Information Systems*, vol. 49, no. 3, pp. 795–833, Dec 2016. Disponible à https://doi.org/10.1007/s10115-016-0918-z

[239] S. N. Narayanan, S. Mittal, et A. Joshi, « Using semantic technologies to mine vehicular context for security, » dans *2016 IEEE 37th Sarnoff Symposium*, Sept 2016, pp. 124–129.

[240] R. Nair, C. Nayak, L. Watkins, K. D. Fairbanks, K. Memon, P. Wang, et W. H. Robinson, *The Resource Usage Viewpoint of Industrial Control System Security : An Inference-Based Intrusion Detection System.* Cham : Springer International Publishing, 2017, pp. 195–223. Disponible à https://doi.org/10.1007/978-3-319-50660-9_8

[241] J. A. Nelder et R. W. Wedderburn, « Generalized linear models, » *Journal of the Royal Statistical Society : Series A (General)*, vol. 135, no. 3, pp. 370–384, 1972.

[242] M. O'Connor et A. Das, « SQWRL : a query language for OWL, » dans *Proceedings of the 6th International Conference on OWL : Experiences and Directions-Volume 529.* CEUR-WS. org, 2009, pp. 208–215.

[243] M. V. M. Oliveira, « Formal Derivation of State-Rich Reactive Programs using Circus, » Thèse de doctorat, Department of Computer Science, Univ. of York, 2005.

[244] A. O. Otuoze, M. W. Mustafa, et R. M. Larik, « Smart grids security challenges : Classification by sources of threats, » *Journal of Electrical Systems and Information Technology*, vol. 5, no. 3, pp. 468 – 483, 2018.

[245] M. R. Palattella, N. Accettura, X. Vilajosana, T. Watteyne, L. A. Grieco, G. Boggia, et M. Dohler, « Standardized Protocol Stack for the Internet of (Important) Things, » *IEEE Communications Surveys Tutorials*, vol. 15, no. 3, pp. 1389–1406, Third 2013.

[246] V. Paxson, « Bro : A System for Detecting Network Intruders in Real-time, » dans *Proceedings of the 7th Conference on USENIX Security Symposium - Volume 7*, série SSYM'98. Berkeley, CA, USA : USENIX Association, 1998, pp. 3–3.

[247] R. Pascanu, C. Gulcehre, K. Cho, et Y. Bengio, « How to construct deep recurrent neural networks, » *arXiv preprint arXiv :1312.6026*, 2013.

[248] N. Pandeeswari et G. Kumar, « Anomaly Detection System in Cloud Environment Using Fuzzy Clustering Based ANN, » *Mobile Networks and Applications*, vol. 21, no. 3, pp. 494–505, Jun 2016. Disponible à https://doi.org/10.1007/s11036-015-0644-x

[249] K. Peng, V. C. M. Leung, et Q. Huang, « Clustering Approach Based on Mini Batch Kmeans for Intrusion Detection System Over Big Data, » *IEEE Access*, vol. 6, pp. 11 897–11 906, 2018.

[250] G. Plotkin, « An operational semantics for CSP, » dans *Workshop on Logic of Programs*. Springer, 1980, pp. 250–252.

[251] D. Papamartzivanos, F. G. Mármol, et G. Kambourakis, « Dendron : Genetic trees driven rule induction for network intrusion detection systems, » *Future Generation Computer Systems*, vol. 79, no. Part 2, pp. 558 – 574, 2018.

[252] A. Patcha et J.-M. Park, « An overview of anomaly detection techniques : Existing solutions and latest technological trends, » *Computer networks*, vol. 51, no. 12, pp. 3448–3470, 2007.

[253] M. S. Parwez, D. B. Rawat, et M. Garuba, « Big Data Analytics for User-Activity Analysis and User-Anomaly Detection in Mobile Wireless

Network, » *IEEE Transactions on Industrial Informatics*, vol. 13, no. 4, pp. 2058–2065, Aug 2017.

[254] K. M. Prasad, A. R. M. Reddy, et K. V. Rao, « BARTD : Bio-inspired anomaly based real time detection of under rated App-DDoS attack on web, » *Journal of King Saud University - Computer and Information Sciences*, 2017.

[255] H. J. Patel, M. A. Temple, et R. O. Baldwin, « Improving ZigBee Device Network Authentication Using Ensemble Decision Tree Classifiers With Radio Frequency Distinct Native Attribute Fingerprinting, » *IEEE Transactions on Reliability*, vol. 64, no. 1, pp. 221–233, March 2015.

[256] S. Parkinson, P. Ward, K. Wilson, et J. Miller, « Cyber Threats Facing Autonomous and Connected Vehicles : Future Challenges, » *IEEE Transactions on Intelligent Transportation Systems*, vol. 18, no. 11, pp. 2898–2915, Nov 2017.

[257] T. Qiu, N. Chen, K. Li, D. Qiao, et Z. Fu, « Heterogeneous ad hoc networks : Architectures, advances and challenges, » *Ad Hoc Networks*, vol. 55, pp. 143 – 152, 2017, self-organizing and Smart Protocols for Heterogeneous Ad hoc Networks.

[258] J. R. Quinlan, « Induction of decision trees, » *Machine learning*, vol. 1, no. 1, pp. 81–106, 1986.

[259] F. Qu, Z. Wu, F.-Y. Wang, et W. Cho, « A security and privacy review of VANETs, » *IEEE Transactions on Intelligent Transportation Systems*, vol. 16, no. 6, pp. 2985–2996, 2015.

[260] S. Ramakrishnan et S. Devaraju, « Attack's Feature Selection-Based Network Intrusion Detection System Using Fuzzy Control Language, » *International Journal of Fuzzy Systems*, vol. 19, no. 2, pp. 316–328, Apr 2017. Disponible à https://doi.org/10.1007/s40815-016-0160-6

[261] G. Reger, « Automata based monitoring and mining of execution traces, » Thèse de doctorat, University of Manchester, UK, 2014.

Disponible à http://www.manchester.ac.uk/escholar/uk-ac-man-scw:225931

[262] D. E. Rumelhart, G. E. Hinton, R. J. Williams *et al.*, « Learning representations by back-propagating errors, » *Cognitive modeling*, vol. 5, no. 3, p. 1, 1988.

[263] L. Rafferty, F. Iqbal, et P. C. K. Hung, *A Security Threat Analysis of Smart Home Network with Vulnerable Dynamic Agents*. Cham : Springer International Publishing, 2017, pp. 127–147. Disponible à https://doi.org/10.1007/978-3-319-62072-5_8

[264] L. Rokach et O. Maimon, « Clustering methods, » dans *Data mining and knowledge discovery handbook*. Springer, 2005, pp. 321–352.

[265] L. Rokach et O. Z. Maimon, *Data mining with decision trees : theory and applications*. World scientific, 2008, vol. 69.

[266] S. Roshan, Y. Miche, A. Akusok, et A. Lendasse, « Adaptive and online network intrusion detection system using clustering and Extreme Learning Machines, » *Journal of the Franklin Institute*, 2017.

[267] M. S. Rahman, M. A. Mahmud, A. M. T. Oo, et H. R. Pota, « Multi-Agent Approach for Enhancing Security of Protection Schemes in Cyber-Physical Energy Systems, » *IEEE Transactions on Industrial Informatics*, vol. 13, no. 2, pp. 436–447, April 2017.

[268] S. Russell, P. Norvig, et A. Intelligence, « A modern approach, » *Artificial Intelligence. Prentice-Hall, Egnlewood Cliffs*, vol. 25, p. 27, 1995.

[269] K. Rina, S. Nath, N. Marchang, et A. Taggu, « Can Clustering be Used to Detect Intrusion During Spectrum Sensing in Cognitive Radio Networks ? » *IEEE Systems Journal*, vol. 12, no. 1, pp. 938–947, March 2018.

[270] E. Rashedi, H. Nezamabadi-pour, et S. Saryazdi, « GSA : A Gravitational Search Algorithm, » *Information Sciences*, vol. 179, no. 13, pp. 2232 – 2248, 2009, special Section on High Order Fuzzy Sets.

[271] M. Roesch, « Snort - Lightweight Intrusion Detection for Networks, » dans *Proceedings of the 13th USENIX Conference on System Adminis-*

tration, série LISA '99. Berkeley, CA, USA : USENIX Association, 1999, pp. 229–238.

[272] J. Rowley, « The wisdom hierarchy : representations of the DIKW hierarchy, » *Journal of information science*, vol. 33, no. 2, pp. 163–180, 2007.

[273] R. Roman, R. Rios, J. A. Onieva, et J. Lopez, « Immune System for the Internet of Things using Edge Technologies, » *IEEE Internet of Things Journal*, pp. 1–1, 2019.

[274] B. Recht, C. Re, S. Wright, et F. Niu, « Hogwild : A lock-free approach to parallelizing stochastic gradient descent, » dans *Advances in neural information processing systems*, 2011, pp. 693–701.

[275] M. G. Raman, N. Somu, K. Kirthivasan, R. Liscano, et V. S. Sriram, « An efficient intrusion detection system based on hypergraph - Genetic algorithm for parameter optimization and feature selection in support vector machine, » *Knowledge-Based Systems*, vol. 134, no. Supplement C, pp. 1 – 12, 2017.

[276] D. E. Rumelhart et D. Zipser, « Feature discovery by competitive learning, » *Cognitive science*, vol. 9, no. 1, pp. 75–112, 1985.

[277] S. Samarah, M. G. Al Zamil, A. F. Aleroud, M. Rawashdeh, M. F. Alhamid, et A. Alamri, « An Efficient Activity Recognition Framework : Toward Privacy-Sensitive Health Data Sensing, » *IEEE Access*, vol. 5, pp. 3848–3859, 2017.

[278] V. Shah, A. K. Aggarwal, et N. Chaubey, « Performance improvement of intrusion detection with fusion of multiple sensors, » *Complex & Intelligent Systems*, vol. 3, no. 1, pp. 33–39, Mar 2017. Disponible à https://doi.org/10.1007/s40747-016-0033-5

[279] K. Salabert, « iASTD, un interpréteur d'ASTD, » *Master Thesis, Department of Computer Science, Univ. de Sherbrooke.*, 2011.

[280] A. S. Sadiq, B. Alkazemi, S. Mirjalili, N. Ahmed, S. Khan, I. Ali, A. K. Pathan, et K. Z. Ghafoor, « An Efficient IDS Using Hybrid Magnetic

Swarm Optimization in WANETs, » *IEEE Access*, vol. 6, pp. 29 041–29 053, 2018.

[281] A. Sarmah, « Intrusion Detection Systems : Definition, Need and Challenges, » SANS Institute, Rapport technique, 2001, https://www.sans.org/reading-room/whitepapers/detection/intrusion-detection-systems-definition-challenges-343.

[282] R. S. Sutton et A. G. Barto, *Reinforcement learning : An introduction*. MIT press, 2018, https://web.stanford.edu/class/psych209/Readings/SuttonBartoIPRLBook2ndEd.pdf.

[283] B. E. Strom, J. A. Battaglia, M. S. Kemmerer, W. Kupersanin, D. P. Miller, C. Wampler, S. M. Whitley, et R. D. Wolf, « Finding Cyber Threats with ATT&CK-Based Analytics, » Rapport technique, 2017, https://www.mitre.org/sites/default/files/publications/16-3713-finding-cyber-threats%20with%20att%26ck-based-analytics.pdf.

[284] H. Shen et L. Chen, « Distributed Autonomous Virtual Resource Management in Datacenters Using Finite-Markov Decision Process, » *IEEE/ACM Transactions on Networking*, vol. 25, no. 6, pp. 3836–3849, Dec 2017.

[285] L. M. Schmitt, « Theory of genetic algorithms, » *Theoretical Computer Science*, vol. 259, no. 1, pp. 1 – 61, 2001.

[286] S. Sabour, N. Frosst, et G. E. Hinton, « Dynamic routing between capsules, » dans *Advances in neural information processing systems*, 2017, pp. 3856–3866.

[287] N. Saxena, S. Grijalva, V. Chukwuka, et A. V. Vasilakos, « Network Security and Privacy Challenges in Smart Vehicle-to-Grid, » *IEEE Wireless Communications*, vol. 24, no. 4, pp. 88–98, Aug 2017.

[288] R. Salakhutdinov et G. Hinton, « Deep boltzmann machines, » dans *Artificial Intelligence and Statistics*, 2009, pp. 448–455.

[289] M. Swarnkar et N. Hubballi, « OCPAD : One class Naive Bayes classifier for payload based anomaly detection, » *Expert Systems with Applications*, vol. 64, no. Supplement C, pp. 330 – 339, 2016.

[290] M. H. Sherif, « Intelligent homes : a new challenge in telecommunications standardization, » *IEEE Communications Magazine*, vol. 40, no. 1, pp. 8–8, Jan 2002.

[291] R. Shittu, A. Healing, R. Ghanea-Hercock, R. Bloomfield, et M. Rajarajan, « Intrusion alert prioritisation and attack detection using post-correlation analysis, » *Computers & Security*, vol. 50, no. Supplement C, pp. 1 – 15, 2015.

[292] S. A. R. Shah et B. Issac, « Performance comparison of intrusion detection systems and application of machine learning to Snort system, » *Future Generation Computer Systems*, 2017.

[293] I. Sharafaldin, A. H. Lashkari, et A. A. Ghorbani, « Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization, » dans *Proceedings of the 4th International Conference on Information Systems Security and Privacy, ICISSP 2018, Funchal, January 22-24, 2018.*, 2018, pp. 108–116.

[294] Q. Sun, H. Li, Z. Ma, C. Wang, J. Campillo, Q. Zhang, F. Wallin, et J. Guo, « A Comprehensive Review of Smart Energy Meters in Intelligent Energy Networks, » *IEEE Internet of Things Journal*, vol. 3, no. 4, pp. 464–479, Aug 2016.

[295] R. R. Smullyan, *First-order logic*. Springer Science & Business Media, 2012, vol. 43.

[296] J. M. Spivey, *The Z Notation : A Reference Manual*. Upper Saddle River, NJ, USA : Prentice-Hall, Inc., 1989.

[297] S. Sicari, A. Rizzardi, L. A. Grieco, et A. Coen-Porisini, « Security, privacy and trust in Internet of Things : The road ahead, » *Computer networks*, vol. 76, pp. 146–164, 2015.

[298] B. Stewart, L. Rosa, L. Maglaras, T. J. Cruz, P. Simões, et H. Janicke, *Effect of Network Architecture Changes on OCSVM Based Intrusion*

Detection System. Cham : Springer International Publishing, 2017, pp. 90–100. Disponible à https://doi.org/10.1007/978-3-319-52569-3_8

[299] F. Sakiz et S. Sen, « A survey of attacks and detection mechanisms on intelligent transportation systems : VANETs and IoV, » *Ad Hoc Networks*, vol. 61, no. Supplement C, pp. 33 – 50, 2017.

[300] H. Sedjelmaci, S. M. Senouci, et N. Ansari, « Intrusion Detection and Ejection Framework Against Lethal Attacks in UAV-Aided Networks : A Bayesian Game-Theoretic Methodology, » *IEEE Transactions on Intelligent Transportation Systems*, vol. 18, no. 5, pp. 1143–1153, May 2017.

[301] S. Shalev-Shwartz et S. Ben-David, *Understanding machine learning : From theory to algorithms*. Cambridge university press, 2014.

[302] O. Singh, J. Singh, et R. Singh, « Multi-level trust based intelligence intrusion detection system to detect the malicious nodes using elliptic curve cryptography in MANET, » *Cluster Computing*, May 2017. Disponible à https://doi.org/10.1007/s10586-017-0927-z

[303] S. Schneider et H. Treharne, « CSP theorems for communicating B machines, » *Formal Aspects of Computing*, vol. 17, no. 4, pp. 390–422, Dec 2005.

[304] V. Stanford, « Pervasive health care applications face tough security challenges, » *IEEE Pervasive Computing*, vol. 1, no. 2, pp. 8–12, April 2002.

[305] K. Saleem, Z. Tan, et W. Buchanan, *Security for Cyber-Physical Systems in Healthcare*. Cham : Springer International Publishing, 2017, pp. 233–251. Disponible à https://doi.org/10.1007/978-3-319-47617-9_12

[306] P. Stone et M. Veloso, « Multiagent systems : A survey from a machine learning perspective, » *Autonomous Robots*, vol. 8, no. 3, pp. 345–383, 2000.

[307] C. Sammut et G. I. Webb, *Encyclopedia of machine learning*. Springer Science & Business Media, 2011.

[308] M. Tavallaee, E. Bagheri, W. Lu, et A. A. Ghorbani", « A detailed analysis of the KDD CUP 99 data set, » dans *2009 IEEE Symposium on Computational Intelligence for Security and Defense Applications*, July 2009, pp. 1–6.

[309] Z. Tian, Y. Cui, L. An, S. Su, X. Yin, L. Yin, et X. Cui, « A Real-Time Correlation of Host-Level Events in Cyber Range Service for Smart Campus, » *IEEE Access*, vol. 6, pp. 35 355–35 364, 2018.

[310] L. N. Tidjon, M. Frappier, M. Leuschel, et A. Mammar, « Extended Algebraic State-Transition Diagrams, » University of Sherbrooke, Rapport technique, 2018, https://depot.gril.usherbrooke.ca/fram1801/TR-25/blob/master/TR-25.pdf.

[311] L. N. Tidjon, M. Frappier, et A. Mammar, « Intrusion Detection Systems : A Cross-Domain Overview, » *IEEE Communications Surveys & Tutorials*, 2019. Disponible à https://doi.org/10.1109/COMST.2019.2922584

[312] L. N. Tidjon, M. Frappier, et A. Mammar, « Intrusion Detection Using ASTDs, » dans *Advanced Information Networking and Applications*, L. Barolli, F. Amato, F. Moscato, T. Enokido, et M. Takizawa, éditeurs. Cham : Springer International Publishing, 2020, pp. 1397–1411.

[313] L. N. Tidjon, M. Frappier, et A. Mammar, « Intrusion Detection Using ASTDs, » dans *Advanced Information Networking and Applications*, L. Barolli, F. Amato, F. Moscato, T. Enokido, et M. Takizawa, éditeurs. Cham : Springer International Publishing, 2020, pp. 1397–1411.

[314] A. Tuor, S. Kaplan, B. Hutchinson, N. Nichols, et S. Robinson, « Deep Learning for Unsupervised Insider Threat Detection in Structured Cybersecurity Data Streams, » *CoRR*, vol. abs/1710.00811, 2017. Disponible à http://arxiv.org/abs/1710.00811

[315] S. Tong, *Active learning : theory and applications*. Stanford University, 2001.

[316] C. Ten, K. Yamashita, Z. Yang, A. V. Vasilakos, et A. Ginter, « Impact Assessment of Hypothesized Cyberattacks on Interconnected Bulk Po-

wer Systems, » *IEEE Transactions on Smart Grid*, vol. 9, no. 5, pp. 4405–4425, Sep. 2018.

[317] M. Vodel et W. Hardt, « Data aggregation and data fusion techniques in WSN/SANET topologies - a critical discussion, » dans *TENCON 2012 IEEE Region 10 Conference*, Nov 2012, pp. 1–6.

[318] P. R. K. Varma, V. V. Kumari, et S. S. Kumar, « Feature Selection Using Relative Fuzzy Entropy and Ant Colony Optimization Applied to Real-time Intrusion Detection System, » *Procedia Computer Science*, vol. 85, no. Supplement C, pp. 503 – 510, 2016, international Conference on Computational Modelling and Security (CMS 2016).

[319] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, et P.-A. Manzagol, « Stacked denoising autoencoders : Learning useful representations in a deep network with a local denoising criterion, » *Journal of machine learning research*, vol. 11, no. Dec, pp. 3371–3408, 2010.

[320] S. Varvaressos, K. Lavoie, A. B. Massé, S. Gaboury, et S. Hallé, « Automated Bug Finding in Video Games : A Case Study for Runtime Monitoring, » dans *2014 IEEE Seventh International Conference on Software Testing, Verification and Validation*, 2014, pp. 143–152.

[321] J. M. Vidal, A. L. S. Orozco, et L. J. G. Villalba, « Alert correlation framework for malware detection by anomaly-based packet payload analysis, » *Journal of Network and Computer Applications*, vol. 97, no. Supplement C, pp. 11 – 22, 2017.

[322] J. M. Vidal, A. L. S. Orozco, et L. J. G. Villalba, « Adaptive artificial immune networks for mitigating DoS flooding attacks, » *Swarm and Evolutionary Computation*, vol. 38, pp. 94 – 108, 2018.

[323] « IEEE 802.15 WPAN Task Group 6 (TG6) Body Area Networks, » http ://www.ieee802.org/15/pub/TG6.html, June 2011, online, accessed 09/27/17.

[324] C. K. Wikle, L. M. Berliner, et N. Cressie, « Hierarchical Bayesian space-time models, » *Environmental and Ecological Statistics*, vol. 5,

no. 2, pp. 117–154, Jun 1998. Disponible à https://doi.org/10.1023/A:1009662704779

[325] J. Woodcock et A. Cavalcanti, « A Concurrent Language for Refinement, » dans *Proceedings of the 5th Irish Conference on Formal Methods*, 2001, pp. 93–115.

[326] C. J. C. H. Watkins et P. Dayan, « Q-learning, » *Machine Learning*, vol. 8, no. 3, pp. 279–292, May 1992. Disponible à https://doi.org/10.1007/BF00992698

[327] K. Wang, M. Du, S. Maharjan, et Y. Sun, « Strategic Honeypot Game Model for Distributed Denial of Service Attacks in the Smart Grid, » *IEEE Transactions on Smart Grid*, vol. 8, no. 5, pp. 2474–2482, Sept 2017.

[328] Webroot, « Threat Intelligence : What is it, and How can it Protect You from Today's Advanced Cyber-Attacks ? » Gartner Inc., Rapport technique, 2014, https ://www.gartner.com/imagesrv/media-products/pdf/webroot/issue1_webroot.pdf.

[329] C. World, *Event Correlation*. IDG Enterprise, 2003.

[330] M. R. Watson, N. u. h. Shirazi, A. K. Marnerides, A. Mauthe, et D. Hutchison, « Malware Detection in Cloud Computing Infrastructures, » *IEEE Transactions on Dependable and Secure Computing*, vol. 13, no. 2, pp. 192–205, March 2016.

[331] W. Wang et M. Zhang, « Tensor Deep Learning Model for Heterogeneous Data Fusion in Internet of Things, » *IEEE Transactions on Emerging Topics in Computational Intelligence*, pp. 1–10, 2018.

[332] X. H. Wang, D. Q. Zhang, T. Gu, et H. K. Pung, « Ontology based context modeling and reasoning using OWL, » dans *Pervasive Computing and Communications Workshops, 2004. Proceedings of the Second IEEE Annual Conference on*. Ieee, 2004, pp. 18–22.

[333] G. Xu, Y. Cao, Y. Ren, X. Li, et Z. Feng, « Network Security Situation Awareness Based on Semantic Ontology and User-Defined Rules for Internet of Things, » *IEEE Access*, vol. 5, pp. 21 046–21 056, 2017.

[334] L. Xiao, C. Xie, M. Min, et W. Zhuang, « User-Centric View of Unmanned Aerial Vehicle Transmission Against Smart Attacks, » *IEEE Transactions on Vehicular Technology*, vol. 67, no. 4, pp. 3420–3430, April 2018.

[335] X.-S. "Yang, « A new metaheuristic bat-inspired algorithm, » *Nature inspired cooperative strategies for optimization (NICSO 2010)*, pp. 65–74, 2010.

[336] X.-S. Yang, *Firefly Algorithms for Multimodal Optimization*. Berlin, Heidelberg : Springer Berlin Heidelberg, 2009, pp. 169–178. Disponible à https://doi.org/10.1007/978-3-642-04944-6_14

[337] J.-H. Yang et C.-C. Chang, « An ID-based remote mutual authentication with key agreement scheme for mobile devices on elliptic curve cryptosystem, » *Computers Security*, vol. 28, no. 3, pp. 138 – 143, 2009.

[338] X.-S. "Yang et S. Deb, « "Cuckoo search via Levy flights", » dans *"Nature & Biologically Inspired Computing, 2009. NaBIC 2009. World Congress on"*. IEEE, 2009, pp. 210–214.

[339] L. Yang, C. Ding, M. Wu, et K. Wang, « Robust detection of false data injection attacks for data aggregation in an Internet of Things-based environmental surveillance, » *Computer Networks*, vol. 129, no. Part 2, pp. 410 – 428, 2017, special Issue on 5G Wireless Networks for IoT and Body Sensors.

[340] X. Yu et M. Gen, *Introduction to Evolutionary Algorithms*. Springer Publishing Company, Incorporated, 2012.

[341] L. Yang, Y. Li, et Z. Li, « Improved-ELM method for detecting false data attack in smart grid, » *International Journal of Electrical Power & Energy Systems*, vol. 91, no. Supplement C, pp. 183 – 191, 2017.

[342] Z. Yi, J. Ma, L. Luo, J. Yu, et Q. Wu, « Improving JavaScript Malware Classifier's Security against Evasion by Particle Swarm Optimization, » dans *2016 IEEE Trustcom/BigDataSE/ISPA*, Aug 2016, pp. 1734–1740.

[343] Y. Yan, Y. Qian, H. Sharif, et D. Tipper, « A Survey on Smart Grid Communication Infrastructures : Motivations, Requirements and Challenges, » *IEEE Communications Surveys Tutorials*, vol. 15, no. 1, pp. 5–20, First 2013.

[344] H. Yu et B. M. Wilamowski, « Levenberg-marquardt training, » *Industrial electronics handbook*, vol. 5, no. 12, p. 1, 2011.

[345] S.-C. Yip, K. Wong, W.-P. Hew, M.-T. Gan, R. C.-W. Phan, et S.-W. Tan, « Detection of energy theft and defective smart meters in smart grids using linear regression, » *International Journal of Electrical Power & Energy Systems*, vol. 91, no. Supplement C, pp. 230 – 240, 2017.

[346] C. T. Zahn, « Graph-Theoretical Methods for Detecting and Describing Gestalt Clusters, » *IEEE Transactions on Computers*, vol. C-20, no. 1, pp. 68–86, Jan 1971.

[347] J. Zhang, Z. Chu, L. Sankar, et O. Kosut, « Can Attackers With Limited Information Exploit Historical Data to Mount Successful False Data Injection Attacks on Power Systems ? » *IEEE Transactions on Power Systems*, vol. 33, no. 5, pp. 4775–4786, Sep. 2018.

[348] Y. Zheng, « Methodologies for Cross-Domain Data Fusion : An Overview, » *IEEE Transactions on Big Data*, September 2015. Disponible à https://www.microsoft.com/en-us/research/publication/methodologies-for-cross-domain-data-fusion-an-overview/

[349] C. Zhong, « Improvements on Graph-based Clustering Methods, » Thèse de doctorat, 2013, http ://cs.uef.fi/sipu/pub/ZhongCaiming-PhDThesis-230813.pdf.

[350] R. Zuech, T. M. Khoshgoftaar, et R. Wald, « Intrusion detection and Big Heterogeneous Data : a Survey, » *Journal of Big Data*, vol. 2, no. 1, p. 3, Feb 2015. Disponible à https://doi.org/10.1186/s40537-015-0013-4

[351] K. Zhang, J. Ni, K. Yang, X. Liang, J. Ren, et X. S. Shen, « Security and privacy in smart city applications : Challenges and solutions, » *IEEE Communications Magazine*, vol. 55, no. 1, pp. 122–129, 2017.

[352] C. Zhang, M. Ni, H. Yin, et K. Qiu, « Developed Density Peak Clustering With Support Vector Data Description for Access Network Intrusion Detection, » *IEEE Access*, vol. 6, pp. 46 356–46 362, 2018.

[353] X. Zhao et W. Zhang, « Hybrid Intrusion Detection Method Based on Improved Bisecting K-Means in Cloud Computing, » dans *2016 13th Web Information Systems and Applications Conference (WISA)*, Sept 2016, pp. 225–230.

[354] Y. Zou, J. Zhu, X. Wang, et L. Hanzo, « A Survey on Wireless Security : Technical Challenges, Recent Advances, and Future Trends, » *Proceedings of the IEEE*, vol. 104, no. 9, pp. 1727–1765, Sept 2016.